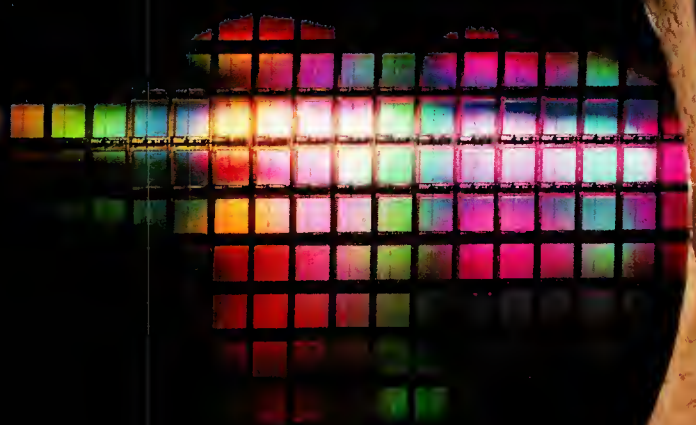
IEEE MICRO

JUNE 1992

Chips, Systems, Software, and Applications

ASSOCIATIVE MEMORIES AND PROCESSORS

MEMORIES THEN AND NOW



IEEE COMPUTER SOCIETY

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

IEEE MICRO

Published by the IEEE Computer Society

June 1992

FEATURES

Cover Image: Mark Gottlieb © 1991, FPG International
Cover design: Chris Martin, Design and Direction

Circulation: *IEEE Micro* (ISSN 0272-1732) is published bimonthly by the IEEE Computer Society, PO Box 3014, Los Alamitos, CA 90720-1264; IEEE Computer Society Headquarters, 1730 Massachusetts Ave., NW, Washington, DC 20036-1903; IEEE Headquarters, 345 East 47th St., New York, NY 10017. Annual subscription: $23 in addition to IEEE Computer Society or any other IEEE society member dues; $39 for members of other technical organizations. This journal is also available in microfiche form.
Postmaster: Send address changes and undelivered copies to *IEEE Micro*, PO Box 3014, Los Alamitos, CA 90720-1264. Second-class postage is paid at New York, NY, and at additional mailing offices. Canadian GST#125634188.

DEPARTMENTS

Call for papers, p. 9, 68; Reader Service cards, p. 64A/B; Membership application, p. 67; Advertiser/Product Index/ Moving Coupon, p. 88; Computer Society information, cover 3

IEEE Computer Society
PO Box 3014, Los Alamitos, CA 90720-1264
(714) 821-8380

* Submit six copies of all articles and special-issue proposals to Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129 Torino, Italy; phone +39-11-556-4044; Compmail: d.delcorso; Internet: delcorso@polito.it; Bitnet: delcorso@itopoli
or
** Ashis Khan, Mips Computer Systems, Inc., 950 DeGuigne Drive, Sunnyvale, CA 94086; Internet: ashis@mips.com.

Author guidelines for article submission appear in this issue, pp. 78–79, and are available from the West Coast office (address at left).

In the mailbag

(LK: liked; DLK: disliked; LTS: like to see)

February 1992

LTS: More multimedia articles.—J.S.V., Miami, FL

LTS: Articles/special edition devoted to tools for DSP-based design—both software (C compilers, code generators) and hardware (DSP ASIC design tools).—R.S.M, Ottawa, Canada

LK: New Products articles and Software Report [on] R&D in Japan; LTS: articles on fuzzy-logic chips, solid-state memory, optical computers and electronics, biosensors.—E.K., Anaheim, CA [This is an interesting list; you will see something on a few of these subjects soon.—D.D.C.]

LK: New Products, Product Summary.—D.J.N., Victoria, Canada

LK: Unix and Am29000, hardware for neural nets [articles]; DLK: ...your new format—articles broken up; [Now articles are kept together; do you mean we should break them again?—D.D.C.]; LTS: articles about sigma-delta codec chips.—M.L.F., San Jose, CA

December 1991

LK: The content of the articles (for the past seven years). [Thanks, also on behalf of previous EICs.—D.D.C.] DLK: The layout. Get rid of the vertical lines separating columns and the lines framing text in each page. Go back to the October 1990 layout. I thought that you would have by now.—V.M., Valley Stream, NY [This is the first time we've received such detailed comments on layout; we change the layout from time to time hoping to improve readability.—D.D.C.]

LK: Fine-grain architecture MDBS [article]; LTS: Unix, CASE, LAN, WAN.—M.C., Hong Kong

LTS: Articles on data compression, especially image data compression. M.A.T., Ankara, Turkey [The October special issue will address this theme.—D.D.C.]

October 1991

DLK: Wide margins....—G.M., Warsaw

DLK: Split articles and editorial comments on readers' remarks. If the editors don't care, we don't care, either. [You have seen that, besides writing comments, we care, and we do something too.—D.D.C., M.E.] LTS: The next EIC from USA. [Me, too; but the nationality does not matter!—D.D.C.]—T.P., Warsaw

August 1991

LK: Clear, concise articles that are well-written; LTS: more details on RISC/CISC chips along with benchmarks. A special issue for a list of articles published to date.—A.C., New Dehli [We publish an annual index (subject and author) in each December issue.—D.D.C.]

LK: Your variety of subjects and your care of up-to-date products; LTS: more about computer networks and parallel processing.—M.A.N., Kuwait [Two requests for networks in the same Mailbag; we shall consider it.—D.D.C.]

Coming Next Issue

IEEE Micro's August issue features articles from European companies such as Mietec Belgium, Philips, SGS-Thomson, and Siemens AG.

Special Features:

- TI's floating-point DSPs as parallel processing building blocks
- On the Edge's discussion of professional registration for electrical engineers

No accolades for *Accolade* court

Richard H. Stern

Oblon, Spivak, McClelland, Maier & Neustadt, P.C.

1755 Jefferson Davis Highway, Suite 400

Arlington, VA 22202

Just as you thought it was safe to go out again and engage in reverse engineering, the San Francisco federal court struck. No more software reverse engineering, the court held, unless you can do it without writing anything down on a paper, disk, or diskette. If you can do your reverse engineering in your head, in RAM, or on an 80-character, 25-line screen, you can stay in business. But put any of the original code or a translation of it into nonvolatile memory and you are liable for copyright infringement.

That is the message in Judge Caulfield's recent (April 3, 1992) preliminary-injunction opinion in *Sega Enterprises, Ltd.* v. *Accolade, Inc.* At the same time as she outlawed ordinary methods of reverse-engineering software, the judge indicated that a manufacturer of a hardware platform is entitled to keep software publishers from marketing software for the platform unless they pay the platform manufacturer for the privilege of doing so. The following discussion focuses on some shortcomings and flaws in the legal analysis that the court offered to support its conclusions.

Sega developed a security system for its video game console, allegedly to prevent counterfeiting of its trademark. (The court's opinion does not explain how the security system stops counterfeiters, and one may well question whether Sega's system can do that or was ever seriously intended to do so.) A "side effect" of the security system is that it keeps out "unauthorized" video game software. "Unauthorized software" is software marketed by a publisher who has not paid Sega a license fee for marketing a game compatible with Sega's hardware. When software that does not co-act with Sega's security system in a particular way is placed in a Sega console, the console will not allow the video game to play; the console rejects the software. Thus, the key issue in the *Accolade* case resembles that involved in the controversy between Nintendo and Atari Games over access to Nintendo's video game console.

Accolade refused to pay Sega for a license, apparently because Accolade believes it is entitled to free access to Sega's hardware platform once the hardware is in the hands of purchasers. The court's opinion, however, found that freedom to be limited by Sega's rights as a copyright owner. The court determined that, when Accolade broke the code for Sega's security system, and placed code in Accolade's software to overcome the security system, it violated several intellectual property rights of Sega.

First, and most important, the reverse engineering in which Accolade engaged involved reproducing a copy of Sega's computer-program code. Apparently, Accolade "unloaded" the object code from Sega's ROMs, thus reproducing a copy (listing) of the object code. (Under the US Copyright Act a copy is an embodiment of a work in a nonvolatile medium such as ROM or printout, as contrasted with RAM or a screen display. Showing code on a screen is not a reproduction of a copy; writing code on paper is.)

Then, Accolade apparently ran the object code through a disassembler, to create more intelligible assembly code from the 1s and 0s of the object code, and printed out the result. That constituted another reproduction of a copy of Sega's work, or perhaps the preparation of a derivative work based on the copyrighted work. (The Copyright Act gives a copyright owner the exclusive right to prepare derivative works based on the copyrighted work. It defines a derivative work as a recasting or transforming of the original version of the work.)

0272-1732/92/0006-0003$03.00 © 1992 IEEE

No more software reverse engineering unless you can do it without writing anything down.

Next, although the court did not make the details of this clear in its opinion, Accolade studied the printout of Sega's assembly code, possibly further transforming or recasting it to make it more intelligible. Accolade then took some of Sega's code (or a derivative-work code based on Sega's code) and placed reproductions of it into the copies of Accolade's own software, to defeat the Sega security system.

Finally, Sega embedded in its security code commands for displaying on the video game monitor Sega's name and a statement (the so-called Sega Message) that the game being displayed on the monitor was produced by or under license from Sega. Accolade copied this code into its software, probably in the course of its efforts to overcome the security system. Accolade either was unable, or did not try, to incorporate the security code and at the same time prevent it from displaying its message on the screen. When Accolade's games were played on Sega consoles, therefore, Sega's name was shown on the screen in association with the games. At the same time, an untrue Sega Message, stating that Sega licensed the software, appeared on the screen. The court considered that to be trademark infringement and unfair competition by Accolade.

The court's opinion does not explain how the Sega security system works or what Accolade did to overcome it. In addition, the court entered a protective order to safeguard Sega's allegedly confidential information, thus further shielding the actual facts from public scrutiny. On the basis of general knowledge of how similar security systems work, however, it is possible to speculate about what the additional factual background may be.

Sega's console contains a small microcomputer that causes display of a video game on a television screen. The console reads information from a ROM in a video game cartridge that a user places into a socket on the console. The stored information comprises data and program instructions used to play the game. In addition to the information for playing the game, Sega probably placed some special code in a location in the cartridge's ROM that the console's microcomputer addresses and reads before beginning to play the game.

For example, Sega may have placed the ASCII code for the letters S-E-G-A in the first several locations in the ROM. It would not be necessary to put the letters S-E-G-A at those locations in the ROM. One could effect the same result with B-E-T-A or B-O-G-U-S or F-L-U-B-G-U-B, or any arbitrary predetermined code. But none of the latter would provide any basis for a claim of trademark infringement.

If the console's microcomputer reads S-E-G-A (or whatever the code is) at that point, it would permit the game play to proceed. (The microcomputer would then also cause the code for the Sega Message to be actuated, so that a display of that message occurred.) But if the code for those letters is *not* found at those locations, the microcomputer may instead shut the console down and refuse to proceed further. We can reasonably infer that *that*, or something like it, occurred in this case. If it did, Accolade would have reacted by putting the code for S-E-G-A in the same locations of its cartridge's ROM as Sega did in its.

So many things are wrong with the court's opinion that it is difficult to focus on any one of them at a time. First, the court misstates holdings in prior reverse-engineering decisions about the subject matter to be tested for infringing similarity to the copyrighted work. The court focused on the copying involved in Accolade's unloading and disassembly of Sega's code. It rejected Accolade's argument that the proper test for infringement was not whether that Accolade code (which the court termed Accolade's "intermediate code") infringed Sega's copyright, but only whether the final commercial code of Accolade infringed Sega's copyright. In rejecting Accolade's argument, the court maintained that the decisions were unanimous that it is proper to enjoin distribution and use of final code that does not infringe, so long as the intermediate code infringed. That is not a correct statement of the law.

Several decisions hold that reverse engineering involving intermediate copies of a copyrighted code is not copyright infringement when the final, commercial version of the code is not substantially similar to the copyrighted code of which the earlier (intermediate) version was a copy. For example, in *NEC Corp.* v. *Intel Corp.* NEC disassembled Intel's microcode, made three versions (Revs. 0, 1, and 2) of the code, and commercially marketed the third version (Rev. 2). The court assumed that Rev. 0 was a direct copy of Intel's copyrighted code, as doubtless it was. Also, NEC's programmer admitted that he disassembled the Intel code and was influenced by what he observed.

Nonetheless, Rev. 2 of the NEC code was not substantially similar to Intel's copyrighted code, and therefore, the court held, NEC was not guilty of copyright infringement. The *NEC-Intel* court clearly considered that the applicable principle of copyright law was that a defendant may legitimately make a direct copy of the plaintiff's work, study it, and then intentionally make enough changes in it to avoid infringement. The *NEC-Intel* court cited nonsoftware copyright decisions to support the

proposition.

The recent decision (now on appeal) in *Computer Associates International, Inc.* v. *Altai, Inc.* also points in the opposite direction from the *Accolade* ruling. In the *Altai* case, defendant Altai marketed a computer program containing some passages of code copied from plaintiff CAI's computer program. After Altai's management learned of the infringement, Altai recalled the infringing programs and substituted rewritten versions from which the offending passages had been excised and into which revised code was inserted. The revised code was dissimilar to CAI's code.

The court held Altai liable to CAI for damages for the initial release, but the court refused to enjoin distribution of the revised program or assess damages for it. Thus, the *Altai* court refused to treat the earlier and later Altai programs as a single unit for copyright liability purposes, even though the later version was a revision of the earlier, infringing version. There is no apparent difference in legal principle between Altai's first commercial version and Accolade's intermediate copy, on the one hand, and Altai's later commercial version and Accolade's commercial version, on the other hand.

The *Accolade* court's claim that the decisions on this legal point unanimously conclude that copyright infringement liability exists when intermediate copying occurs is simply wrong. There is ample precedent for the opposing rule that an accused work infringes a copyrighted work *only* if the accused work contains a substantial amount of protected expression taken from the copyrighted work, without regard to earlier versions of the accused work.

Finally, the court said that Accolade should have acted under the protected form of reverse engineering allowed by the Semiconductor Chip Protection Act (SCPA), instead of disassembling Sega's code. According to the court, "Accolade could have 'peeled' the microchips as set forth in [17 U.S.C.] § 906." Then it would not have infringed Sega's copyright. That is wholly erroneous. The reverse-engineering provisions of the chip law do not immunize copying of computer code against liability for copyright infringement; they merely immunize from SCPA liability some forms of copying of chip layouts.

If Accolade had followed the court's advice about peeling the chip, it would have been just as liable for copyright infringement as the court found it to be for disassembling the code. There appears to be no feasible way to reverse-engineer a security code of this type without making an intermediate copy to analyze. Once Accolade peeled the ROMs, it would have needed to write down the information thus discerned to make any sense out of it. The court simply has no idea of how you reverse-engineer software, or what an engineer does when peeling a chip.

The issue never addressed in the *Accolade* opinion is this. Is the copyright system properly utilized as a means of keeping software publishers out of hardware platforms if they do not agree to pay the hardware sellers for the privilege of selling software that can run on the hardware? This fee is exacted as a royalty for the use of code needed to overcome the hardware sellers' security systems. Here, the video game security system has one purpose: It does not make the Sega video game console run faster, display better pictures, or provide increased enjoyment to game players. It does not do anything but keep out video game software that lacks the code the security system requires to let the game software "pass GO."

Sega takes the position that it is entitled to make software publishers pay it to sell games that can be played on the Sega console. Presumably, that price is reflected in what end users have to pay for games. This is a new variation on selling razors cheaply and pricing compatible razor blades high. In other words, for this court, the old *A.B. Dick* decision rides again. In this 1912

The court has no idea of how you reverse-engineer software, or what an engineer does when "peeling" a chip.

decision (later overruled), the Supreme Court held it permissible for A.B. Dick to license its patented mimeograph machines on the condition that customers must buy ink and stencils from it or its designee.

There is no patent or copyright infringed when an end user of video games places an "unauthorized" video game into a Sega console and plays it. Or there was none until Sega installed its security system. What is the legal interest that the copyright law is protecting in this case? What does protecting this interest do to the interests of end users (the public)? Is the interest one that we want to protect? Does Sega's conduct further the goals of the copyright system? Or is it a misuse—or at least not an equitable use—of copyright law?

The *Accolade* court's opinion left these issues unaddressed, certainly unresolved, and apparently unrecognized as having any bearing on whether an injunction should issue. The court brushed aside any policy issues, stating that Congress had already resolved all policy questions in favor of copyright owners and against copyists.

That is simplistic. It attributes to Congress an intent where none was ever expressed. Certainly, Congress never stated that its intent was to resolve all cases, in which interests must be balanced, in favor of plaintiffs and against

defendants—as the *Accolade* court seems to suggest. It is equally plausible that Congress made a decision in favor of competition and free access to systems and other ideas by codifying *Baker* v. *Selden* into section 102(b) of the statute. It provides: "In no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work."

A preferable legal approach to this controversy would have focused on interests of end users. A purchaser of a Sega console purchases it for use and enjoyment, specifically for playing video games. The purchaser never agreed to use the consoles only with Sega's products or those of its licensees. The purchasers of these consoles thus have a right, as property owners, to use the consoles to play whatever video games they see fit.

Sega attempted to circumvent that right. It placed a device in the consoles to lock out video game cartridges unless their ROMs contained copyrighted computer-program code (in the example given earlier, the code for the letters S-E-G-A), which would "unlock" the security system.

Enforcement of the copyright in this computer-program code both fails to further the copyright law's goal of promoting the progress of human knowledge, and interferes with the exercise of the property rights of purchasers of the products containing the copyrighted code. It should not, therefore, be considered a copyright infringement for purchasers of Sega consoles to take or use the copyrighted Sega security code to get the benefit of their purchases of consoles from Sega. Sega should be considered to be estopped from challenging the purchasers' taking from Sega what is in effect, the key to a lock that Sega secretly inserted into the purchasers' personal property.

Finally, what bearing does such a theory of consumers' rights have on the position of Accolade as an infringer of Sega's copyright? The practicality of console purchasers' situations must be considered. Consumers are in no position to make their own video game cartridges. As a practical matter, they must purchase them from manufacturers of such equipment. Otherwise, the purchasers' rights as just described would be illusory, as would be, also, for example, those of car owners to buy repair parts, sponge-mop owners to buy replacement sponges, and personal-computer owners to buy new software.

It is a familiar principle of law that when A has a right or privilege that can be vindicated only by means of the assistance of B, A's privilege transfers to B to shield B from liability when assisting A. Under that principle, therefore, the court should have found Accolade's conduct privileged. Software sellers should not be liable for using a hardware seller's code to help end users vindicate their right to buy software for their own machines from whomever they please, without having to pay an additional fee to the hardware seller for the privilege of using their own property.

The purchaser never agreed to use the consoles only with Sega's products or those of its licensees.

The *Accolade* decision is a sorry example of what happens when judges who know nothing about software get their hands on a software controversy. The decision is a loud argument for creating a specialized court to handle software cases—or at least for referring such cases to technical experts who will not make gross blunders because they cannot understand the technology. If this decision stands, it will set software progress back by decades. Moreover, it will hand the software business over to offshore developers by making it illegal to reverse-engineer software within the United States. This is truly a new low.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 177 Medium 178 High 179

Send information for inclusion in Micro News one month before cover date to Managing Editor, IEEE Micro, *PO Box 3014, Los Alamitos, CA 90720-1264.*

From desktop to palmtop

Ware Myers, Contributing Editor

Desktop computers have no tight limits on size, weight, power consumption, memory capacity, disk capacity, screen size, or brightness. In general, they can be designed to human size. Keys fit human fingers. The keyboard can be large enough to accommodate key sets beyond the basic qwerty set. Additional sets may include function keys, a cursor-movement pad, and a number-entry pad. The cathode-ray tube, though relatively large, heavy, and power-consuming, provides a page-size color screen bright enough to read in ordinary room illumination.

The specifications of most desktops are in the range of

- a keyboard of 14.5 × 5 inches,
- a CRT screen of 9 × 7 inches or even up to two 8.5 × 11-inch pages,
- screen resolution of 640 × 480 pixels, or 24 lines of 80 characters,
- a hard-disk drive of between 40 and 120 Mbytes, and
- an internal memory of 1 to 8 Mbytes standard, expandable to 32 Mbytes.

Notebook computers. The notebook computer represents the effort to get the personal computer as small and light as possible while still large enough to match the keyboard and screen with a human scale. Thus, the typical notebook measures 11 × 8.5 × 1.75 inches. At this size it can have a standard qwerty keyboard and a 640 × 480-pixel screen (24 lines of 80 characters). It weighs from 5 to 7 pounds. A slightly larger size, on the order of 13 × 11 inches, is sometimes called the laptop.

These computers run standard operating systems, so nearly all of the 40,000 or more available application programs will run on them. Of course, most users want regular access to only a few programs—word processing, spreadsheet, calendar and schedule, expense reporting, database, telephone numbers, and communications. Many marketers make a point of integrating notebook software by building in common capabilities such as these. About 60 companies market 100 laptop and notebook computers.

Notebook computers have many deficiencies, of course, compared to desktops. One is their general lack of electrical power. To make them usable beyond the range of an extension cord, they run on batteries. Consequently, the batteries run only two or three hours before they need recharging. Since notebooks are usually used intermittently, this charge can often last an entire working day. A major improvement for operating life comes from a new microprocessor that can stop the clock. Toshiba claims it doubles battery operation time.

The limited power, as well as weight and size restrictions, necessitate the use of liquid crystal displays rather than the much brighter cathode-ray tubes. LCDs have had three drawbacks: They are not very bright, the screens have been on the small side, and they are black and white. Brighter and larger active-matrix LCDs, which can provide color, are coming into use, but at higher prices. Besides cost, a major problem with LCDs is power consumption. Currently, some companies that sell color notebooks do not even specify operating time on batteries–a bad sign!

Size restrictions usually limit the keyboard to the basic qwerty set, a disadvantage of some consequence in some applications. For example, accountants, statisticians, and other number-using

0272-1732/92/0600-0007$03.00 © 1992 IEEE

professionals feel the lack of number entry keys.

Palmtop computers. Designers have been able to put all the electronics of a computer system on a few chips for some time. Calculator makers have built physically small versions of computers with limited input and output for years. When the size of a personal computer is reduced well below the notebook level, it becomes a palmtop computer, also called a pocketable, hand-held, or picocomputer.

The problem is not in making the computing elements small, but in enabling the user to input data to the computer and observe the output on his or her own size scale. Input and output capabilities in the form of keyboards and screens suited to human scale occupy space, add weight, and draw electric power.

Keyboard input is the typical entry method to desktop and notebook computers. But the palmtop size is considerably smaller than a full-size keyboard. One solution to its entry problem is a very small keyboard. Psion, Inc. took that path. The Psion Series 3 has 58 keys in a qwerty pattern within outside dimensions of only 6.5 × 3.3 × 0.9 inches. It weighs 10 ounces. A base model costs $425, but options can double that figure.

Hewlett-Packard, a long-time maker of calculators, has also taken that path with its HP 95LX. It has not only a reduced-size qwerty keyboard, but an adjoining numeric keypad. It weighs 11 ounces and costs $699, but options can run the price up.

Poqet Computer Corporation attempted to straddle both worlds with a 1-pound unit whose keyboard is 80 percent of standard size (*IEEE Micro*, Feb. 1990, p. 9).

A user can enter data into these small keyboards in hunt-and-peck fashion, but cannot touch-type. For people in the field who are making only a few entries now and then of an order or inventory status, the small-keyboard solution may be satisfactory. A lot of people in field occupations don't touch-type anyway.

Many potential users of palmtop computers don't like keyboards in any case. They would rather handwrite their entries. A new category of pen-based computers is reaching this market. In general, users have to print the letters within marked blocks. Unfortunately, current recognition logic correctly identifies only about 98 percent of characters and perhaps 90 percent of words. When the logic recognizes (or thinks it recognizes) the handprinted letter, it substitutes a printed character. The user has to watch this feedback on the small screen and keep making corrections.

Handwriters are accustomed to two sizes: tablet (or clipboard) and stenographic notebook. Not surprisingly, the screen on which the stylus writes is often near these sizes. For example, the exterior dimensions of the Grid Pad/PC from Grid Systems are 9.2 × 12.4 × 1.4 inches. It weighs 4.5 pounds and costs $2,595. IBM last April announced a tablet-size Think Pad weighing 6 pounds. Initial deliveries are limited to software developers.

Hewlett-Packard's 95LX

The Momenta 1/40 is a transition system for those who want to handwrite in the field and key-enter in the office. Its base unit is tablet-size (11.5 × 12.5 × 2.5 inches), 6.5 pounds, but a separate keyboard can be plugged into it. It costs $4,995.

The ultimate entry method would be voice. The hardware and software market in voice recognition is already at several hundred million dollars a year, mostly in large systems. Apple Computer plans to offer voice recognition as a Macintosh option. A user would be able to enter commands by voice.

Output from a palmtop computer presents a problem similar to input. The screen is very small, compared to the notebook or desktop. The Psion Series 3 LCD screen, for example, provides only 240 × 80 pixels, or eight rows of 40 characters. This amount of display is not sufficient for extensive composition or a spreadsheet of any size, but it is adequate for entering characters pertaining to orders or inventory. The HP 95LX is better, containing 16 rows of 40 characters. A field worker might find these sizes satisfactory for composing short notes to the office. A traveling executive could make a few changes in material sent out over electronic mail for approval.

Mass storage presents other size, weight, and power problems. Hard disks have been getting steadily smaller for years and have now reached 2.5 inches. The HP 95LX relegates mass storage to the user's desktop PC by means of the HP F1001A connectivity option. The mass-storage solution may be flash memory cards. The Psion Series 3 accommodates up to 2 Mbytes of flash memory, which simulates A and B disk drives.

Last April Intel introduced a 20-Mbyte credit-card-size flash memory card priced at about $600. Earlier 4-Mbyte cards had cost $1,200, too high to compete with disk drives. Using a flash card rather than a hard disk, designers can reduce the size, weight, and power consumption of future palmtop computers.

Software. Notebook-size computers generally operate on conventional operating systems, such as MS-DOS. Pen-based computers, however, face a different set of requirements that have led several vendors to develop operating systems specially for them. Go Corporation began shipping Pen Point to software developers in the spring of

1991 and recently listed 50 application programs from 22 software vendors. This last spring Microsoft issued an early version of its Windows for Pen, primarily for software development.

Compared to the tens of thousands of application programs available on personal computer operating systems, a few dozen is a thin diet. Expansion of sales of pen-based systems will undoubtedly go hand-in-hand with expansion of the application programs available.

The Psion Series 3 is DOS-compatible, but its built-in applications provide a glimpse of what one manufacturer regards as the central needs. They are word processor, outlining, database, telephone dialer, world feature map, agenda, calendar, planner, appointments diary, time and alarm, to-do list, notes, communications, and scientific calculator. The user selects an application merely by pressing an icon on a touch screen. The HP 95LX is also DOS-compatible. A ROM contains its built-in software, which includes Lotus 1-2-3.

One company president who travels frequently has used the HP 95LX for about a year. "With its integrated built-in software, it will do 98 percent of everything you want to do with a computer," he told me. "You can keep all the applications you are using open at the same time and shift back and forth with a single keystroke. The ability to move between the calculator and Lotus 1-2-3 is incredibly good. You can cut and paste between applications very easily.

"Because of the small keyboard, it isn't much of a word processor, but that is really its only major drawback," he continued. "I use it for short notes and memos and it is fine for that. You can't touch-type with it, but with two-finger pecking it is adequate.

"The fact that HP made it easy to hook up to your PC makes its shortcomings go away," he noted. "You can type and revise text on the PC where it is too tedious on the small machine. You can also file on the PC. It has been possible to do these things for some time, but not nearly as transparently as HP has made it."

A market beckons. What can we foresee of the palmtop market at this time? First, are there any potential users we can exclude? Well, yes, most desktop computer users will probably stay with that size. Many executives and professionals, though they work at a

The Psion Series 3

desk most of the time, do not type well. They will like pen-based or voice-input computers when they appear. Moreover, their small size and weight will be convenient when they travel, as many of them often do.

Similarly, most users of notebooks will stay with that size, because input and output are better than the palmtops offer, at least until voice I/O arrives.

Second, that still leaves a vast market of professionals that do not work at desks, but in cars or trucks, in clients' offices (consultants, salesmen, service men, and so on), in fields, warehouses, plants, and hospitals. Some occupations work standing up, such as medical personnel making rounds. They can punch a few keys with one finger, or handwrite brief entries. Some early users of pen-based computers use them to take notes in meetings. They say taking notes by hand distracts the other participants less than typing.

As word of their myriad uses spreads and additional uses are discovered, palmtop computers will undoubtedly reach a vast market during this decade—up in the tens of millions of units.

continued on p. 80

Guest Editor's Introduction
Associative Processors and Memories

Karl E. Grosspietsch

German National Research Center for Computer Science

Since the early days of computer design in the 1940s, the scientific community has discussed alternatives to the traditional von Neumann computer architecture. The von Neumann architecture had its clear merits under the technology restrictions of the early phase of computer development. Soon, however, the research community recognized the trade-offs of this approach:

- Data processing is restricted to the very rigid scheme of a predefined instruction stream, mostly independent of intrinsic properties of data. Only the use of Boolean variables to split programs into alternative paths provides some flexibility.
- The clear distinction between the functions of data storage (in memory) and data processing (in the CPU) necessitates data transfers between memory and CPU whenever data are changed.
- The classical von Neumann style of programming is strictly sequential. Parallelization of programs written in the "imperative" von Neumann style is possible but creates nontrivial synchronization problems.

In recent decades, researchers have proposed various architectures to overcome these trade-offs. Many approaches circumvented the sequential limitations of von Neumann machines by linking several together for parallel data processing. These processors can operate under one instruction stream: the single-instruction, multiple-data (SIMD) principle. Or they can operate under decentralized control according to the multiple-instruction multiple-data (MIMD) principle used in distributed systems.

Some researchers tried radically different directions, for example, dataflow machines or architectures for functional programming. Such approaches had interesting and promising properties, but their transfer into commercial products usually failed. Their development costs (including the production of complete software application packages) were too high to compete with the existing computer generations.

The main interest has focused on "evolutionary" solutions compatible with the traditional way of computing and executed by computer components modularly added to existing hardware systems. Thus, coprocessor concepts have found increasing importance.

In the last decade, two main conditions supported such evolutionary development:

- Demand is growing for more "intelligent" computer systems, that is, subsystems within

0272-1732/92/0006-0010$03.00 © 1992 IEEE

the hardware system that automatically take over from the end user a growing spectrum of tasks. Examples are picture processing and preprocessing of sensor data.
- Advanced high-integration technologies, such as very large scale integration (VLSI) and wafer-scale integration (WSI), now enable realization of innovative complex architectures on small chip areas at comparatively low costs.

In this context, systems for associative (a synonym is content-addressable) storage and processing of data are an interesting architectural approach for inherently smarter computer systems. Computer architects have discussed content-addressable memories, or CAMs, since the early 1950s.[1] Because of technological restrictions at that time, such approaches seldom moved beyond laboratory prototypes. Now high-integration technology for the first time permits the implementation of such memories with acceptable capacity/cost ratios.

Moreover, functional integration of processing logic within the CAM structure appears possible. Interesting applications emerge for associative processors that can process data with respect to their properties. Such systems comply well with the standard imperative programming paradigm of today's classical computers.

This special issue of *IEEE Micro* presents discussions of the state of the art and the recent progress in the field of associative processors and memories. Because it is beyond the scope of this issue, we do not consider the related field of memory or processor structures based on neural net approaches. (These are also often called "associative." For a survey of that area, see *IEEE Micro*'s December 1989 issue.[2])

This issue comprises five articles from industrial and academic research. The introductory article surveys development in the area. The second article by I.N. Robinson describes a special approach for associative memory. The pattern-addressable memory (PAM) is an architecture especially tailored to efficient retrieval and processing of patterns stored as symbolic data structures. The subsequent article by F.P. Herrmann and C.G. Sodini addresses a special associative processor for machine vision applications. In the fourth article, R. Storer, M.R. Pout, A.R. Thomson, E.L. Dagless, A.W.G. Duller, A.P. Marriott, and P.J. Hicks discuss a similar application field.

A principal problem of CAM structures is the connection of several CAM chip components: The coupling between these components by signal lines must be much stronger than in RAMs. The final article in this issue by T. Moors and A. Cantoni considers general strategies for such cascading of content-addressable memories.

A single issue could not accommodate all the excellent contributions that were submitted on this topic. A future issue of *IEEE Micro* will present additional articles on associative processors and memories.

References

1. T. Kohonen, *Content-Addressable Memories*, Springer-Verlag, Berlin, 1980.
2. D. Del Corso, K.E. Grosspietsch, and P. Treleaven, eds., "Silicon Neural Networks," special issue, *IEEE Micro*, Vol. 9, No. 6, Dec. 1989.

Karl E. Grosspietsch is a researcher at the German National Research Center for Computer Science, Gesellschaft fuer Mathematik und Datenverarbeitung (GMD), in St. Augustin, Germany. He joined GMD in 1974. His main activities comprise research in the fields of computer architecture, fault tolerance, and VLSI design.

Grosspietsch studied computer science at the University of Hamburg, where he received his diploma. He received a PhD in computer science from the University of Bonn. He is a member of the German computing society, Gesellschaft fuer Informatik, and serves as Associate Editor-in-Chief of *IEEE Micro*.

Address questions about this article to K.E. Grosspietsch at the German National Research Center for Computer Science (GMD), PO Box 1316, D-5205 St. Augustin, Germany; grossp@gmdzi.uucp.

Associative Processors and Memories: A Survey

Because of declining hardware prices, associative (content-addressable) architectures are again gaining importance, especially in artificial intelligence and database applications. This survey introduces the classical content-addressable memory and explains its realization at the transistor level. Then it describes some unorthodox CAM approaches, discusses associative processor systems, and classifies current approaches.

Karl E. Grosspietsch

German National Research Center for Computer Science

Computer architects have discussed associative systems since the early days of computer design. Associative system design features provide alternatives or additions to the classical von Neumann machine architecture.[1,2] In this brief survey of recent developments in the field, let's first consider the functional structure of a classical content-addressable memory (CAM) and its realization at the transistor level. Then we can look at some unorthodox approaches for CAMs, discuss associative processor systems, and classify the existing approaches in this field.

In this article, "associative" is synonymous with "content addressable." In some contexts, the term is linked with memory or processor structures based on neural net approaches, which we won't consider here.

Associative memories

In contrast with access via memory addresses in a conventional RAM, in a CAM the system accesses the content of a word cell by a comparison with a given search argument.[1] Functionally, the word cells of a typical CAM consist of two parts:

- a *search key field* whose contents can be compared with the search argument by some associative logic, and
- the *data field* built of conventional RAM bit cells.

If the search key field of a word cell i ($i = 0, ..., w - 1$) matches the search argument, a hit line emanating from the search key field activates the bit cells of the data field of word cell i (see Figure 1). Let k denote the length of the search key field, and l denote that of the data field; the entire word length is $n = k + l$. For $l = 0$ we have the special case of a CAM in which each bit slice has associative search logic.

When the system compares a search pattern with the search key fields, it loads the search pattern into the search argument register (SAR). It performs the subsequent comparison for each w word cell of the CAM concurrently. Inside each word cell, the comparison is presumed to be carried out bitwise in parallel. If the key field of a word cell i equals the given search argument, a hit signal results and is stored in cell i of the hit register (see Figure 1).

0272-1732/92/0600-0012$03.00 © 1992 IEEE

A second input register, the mask register, may mask the comparison with the contents of the SAR. If a bit j ($j = 0, \ldots, k-1$) of this mask register is set to 1, in all bit cells of the corresponding bit slice j the comparison is not carried out. Instead, these cells always generate a local hit signal. In this way, the mask register can reduce the number of effective bits in the search key field.

This memory structure allows multiple hits within the CAM. In the case of a write operation, the same bit pattern can be written from the memory data register (MDR) into all word cells found via their search key fields. In the case of an associative read operation with more than one hit, the system must serialize the output of the words found. A priority logic does this by selecting one word cell from the set of word cells found—for example, the one with the highest internal address i, which is given by the bit position of the hit signal in the hit register. If word cell i has been selected, the corresponding output line of the priority logic is set to 1; all other output lines have the value 0.

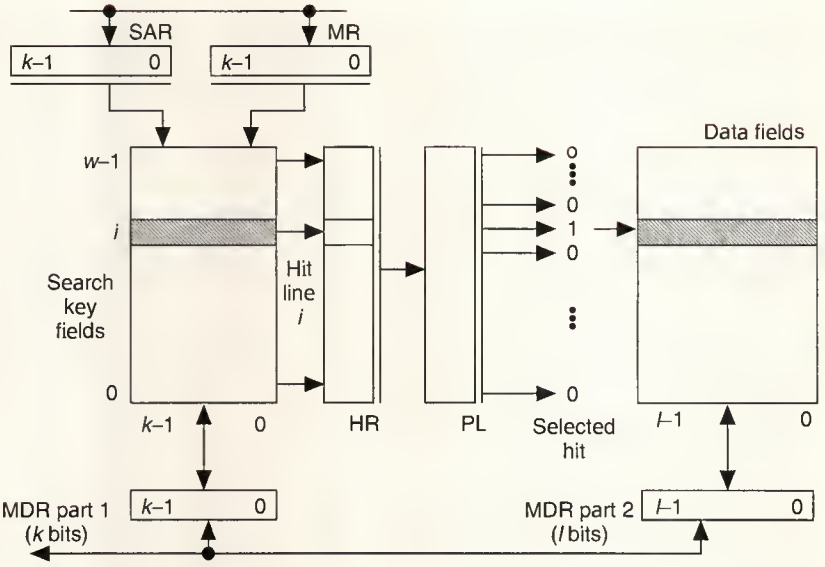


Figure 1. Architecture of a classical CAM. (HR indicates hit register; MDR, memory data register; MR, mask register; PL, priority logic; SAR, search argument register; and shaded areas, the two parts of word cell i).

Some approaches for CAMs also provide an individual "masked" state for every CAM bit cell. Therefore, such a memory (also called functional memory) requires a bit cell with at least three different storage states: 0, 1, and don't care. For a more detailed discussion of this aspect, see the contributions of Herrmann and Sodini and Moors and Cantoni in this issue, pp. 31–41 and 56–67.

Realization of CAM bit cells

One way to build a CAM bit cell is to extend the classical static RAM flip-flop cell.[1] Figure 2 shows such an approach. In addition to the six transistors of the flip-flop, three other transistors implement the match logic. Writing a 1 (0) is performed as in a RAM. The pass transistors T1 and T2 are opened via the word line. The bit line Bit is set to 1 (0), whereas the complementary line Bit′ is driven to the inverted signal of Bit. So transistor T6 (T5) is conducting, and T5 (T6) is nonconducting, thus representing the storage of the written value.

The comparison logic is simply realized by two pass transistors T7 and T8. If the system compares the cell's content with a given search bit s, the match line is first precharged to high. Then line Bit is set to the inverted search value s' (and, correspondingly, Bit′ to s). If a 1 was stored in the flip-flop so node 1 is high, transistor T7 propagates the value 0 from line Bit to the gate of T9. So this transistor remains nonconducting, and the match line is not discharged to ground.

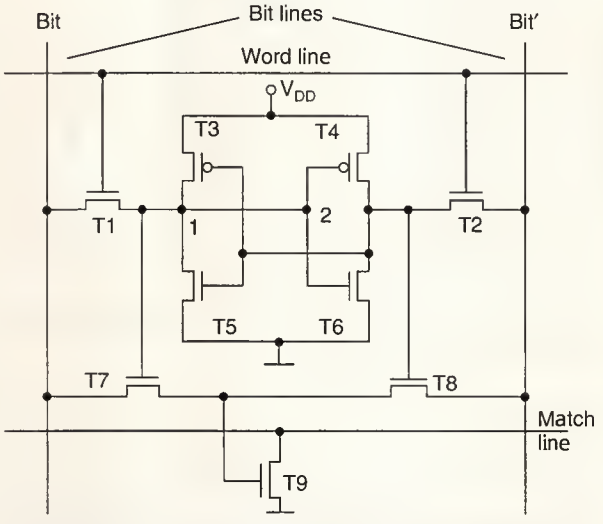


Figure 2. Structure of a nine-transistor CAM bit cell.

Any mismatch (in our example, a coincidence of a signal 1 of line Bit with the high value of node 1) opens T9, discharging the match line. Driving both bit lines to 0 masks the bit cell.

A similar approach simply uses four additional pass transistors to combine nodes 1 and 2 by negated XORs (see Fig-

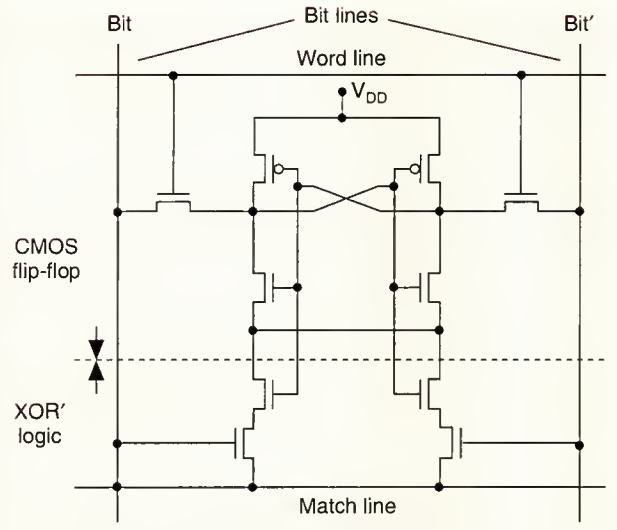


Figure 3. Ten-transistor CAM bit cell.[3]

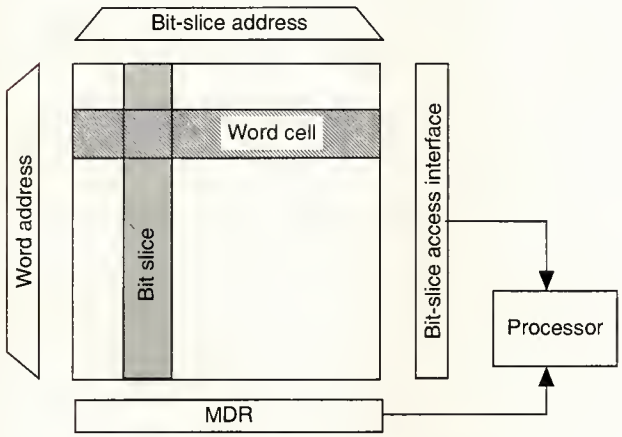


Figure 4. Orthogonal memory.

ure 3). The output of these XOR's is wired to the match line. In this 10-transistor cell, driving the line Bit to the noninverted search value s (and Bit' to $\bar{s}$) carries out a match against a stored bit. A mismatch again causes the discharge of the precharged match line; applying 0s to both bit lines masks the bit cell. On the basis of this circuit architecture, Kadota et al.[3] developed a CAM chip comprising 256 words of 32 bits; that is, the entire chip had a storage capacity of 8 Kbits. To avoid resistance-capacitance delays and minimize access time, they used low-resistance double-layer metallization techniques to lay out the bit and match lines.

In a comparable approach, Ogura, Yamada, and Nikaido[4] developed a CAM chip with a 4-Kbit capacity. It consisted of 128 words of 32 bits. Apart from retrieval functions, this chip could also write the same data into multiple words in parallel. Two years later, a more advanced solution for a 20-Kbit CAM evolved from this implementation.[5] The chip had a bit cell array containing 512 words of 40 bits and functional blocks for bit, word, and address operations.

These chips were more or less research prototypes. In 1987, Advanced Micro Devices introduced the first commercial version of a VLSI CAM chip, the Am95C85. It stored 1 Kbit of data. The memory could respond to an 8-bit key in about 10 µs. In 1989, the company followed up with the 12-Kbit Am99C10 chip. This chip uses a 48-bit-wide key that can be compared in parallel with 256 words. An intended application area is address management in local area networks.

Some approaches realize CAMs by extending dynamic RAM cells.[6] DRAM techniques achieve considerably larger bit capacities per chip because they have fewer transistors per bit cell (one or three transistors per cell). However, with these techniques either leakage currents or destructive read operations necessitate refresh operations. Because the system reads several bit cells simultaneously in a CAM, a destructive read operation as used in conventional dynamic RAMs is not possible. Also, data must be stored so a mismatch cannot destroy them. This can be accomplished by storing the data on the gates of two transistors. (For a detailed discussion of ways to implement such dynamic CAM cells, see Herrmann and Sodini's contribution in this issue.)

Unorthodox CAM approaches

An approach that differs significantly from the classical CAM structure is the orthogonal memory recently proposed by Kokubu, Kuroda, and Furuya.[7] (Batcher previously realized a similar structure in the Staran machine.[8]) The orthogonal memory does not provide comparison logic in the bit cells. Thus, the bit cell is quite similar to that of a RAM. However, two additional pass transistors permit the cell contents to be read out to the word line (see Figure 4). Apart from the normal random-access mode, the system can access an entire bit slice concurrently in a second memory mode and compare the slice outside the cell array, bitwise in parallel with the search pattern. Thus, it can easily perform a word-parallel, bit-sequential associative search.

Another very unorthodox approach recently proposed by Tavangarian[9] and Waldschmidt[10] is the associative random-access memory (ARAM). It is also called "location-addressable" associative memory and is realized mainly by a very simple change of the usual RAM decoder. In addition to the Nand gates $G_0 \ldots G_{w-1}$, we have the additional Nand gates $A_0 \ldots A_{2^m-1}$ ($m = ld\ w$ being the length of the memory address), which combine the input address with a mask pattern from an additional decoder mask (see Figure 5). If the mask is zero, the extended decoder functions like a normal 1-out-of-w decoder.

Otherwise, it carries out a multiaccess to all cells with addresses matching the unmasked bits of the input address.

The corresponding memory array is—in the simplest approach—formed by exactly one bit slice. Associative processing is then based on the following rule: The system stores an m-bit pattern $b_0 \ldots b_{m-1}$ in memory by setting a 1 as a flag to the cell with the address $b_0 \ldots b_{m-1}$. Analogously, the system searches for the pattern by checking whether the corresponding bit cell stores a 1 or a 0. The advantage of this architecture is the easy cascadability of the memory (for the bit cells themselves, standard RAM technology can be exploited). Moreover, the data stored in the ARAM are structured in ascending order. The data organization supports search operations such as queries about whether bit patterns larger or smaller than a given bound or patterns within a given interval of binary numbers are stored in the ARAM. The trade-off is that increasing the bit length of the patterns to be stored in the ARAM by i bits requires an increase in address space by a factor 2^i. So ARAM application is confined to data with relatively short bit length.

Associative processor systems

The notion of content-controlled access to data can be generalized to the processing of data: Not only are data retrieved according to their properties, but their updating or deleting is organized in that way. Such an associative processor system can, for example, be realized by placing a CAM as a data memory inside an environment of some processing logic, either a conventional host monoprocessor or arrays of processing elements. Several approaches for associative processor systems are based on assigning some processing logic to each CAM word cell, in addition to the priority logic that is also necessary. Some approaches also consider the integration of processing logic directly within the CAM bit cells.

Researchers investigated such processor systems already in the early days of computer science in the 1950s and 1960s. Probably the most well-known early approach for building an associative processor system was Batcher's Staran computer.[8] This system used a memory with access capabilities similar to the orthogonal memory: Apart from normal word access, it also enabled access to entire bit slices. Moreover, it provided access features between these two extremes: In a mixed mode, a regular diagonal pattern of bit groups was accessible throughout the memory.

Foster's book[11] and Thurber and Wald's and Yau and Fung's survey papers[12,13] provide good overviews of early approaches. Usually, these approaches could not transfer to the market because of high hardware costs.

AI applications

Several approaches for developing application-specific CAM architectures to support artificial intelligence features[14] have recently been reported. Kogge et al.[15] at Syracuse University developed a CAM tailored to specific AI applications. These applications use some form of If-Then rule programming. In languages that mainly use production rules, the system compares data against If parts until the arguments are satisfied. The Then parts usually indicate how the data are to be changed. Other more deductive languages such as Prolog perform matches between a goal—the truth value of which is usually not known—and the Then part of a rule.

Kogge et al. developed a VLSI coprocessor together with a CAM to work as a hardware accelerator for such tasks. The CAM is based on a special 10-transistor bit cell. For the applications considered, the flexible cascading of CAM chips is essential because of additional words and increased word length. In the approach, the basic word length is 32 bits. An

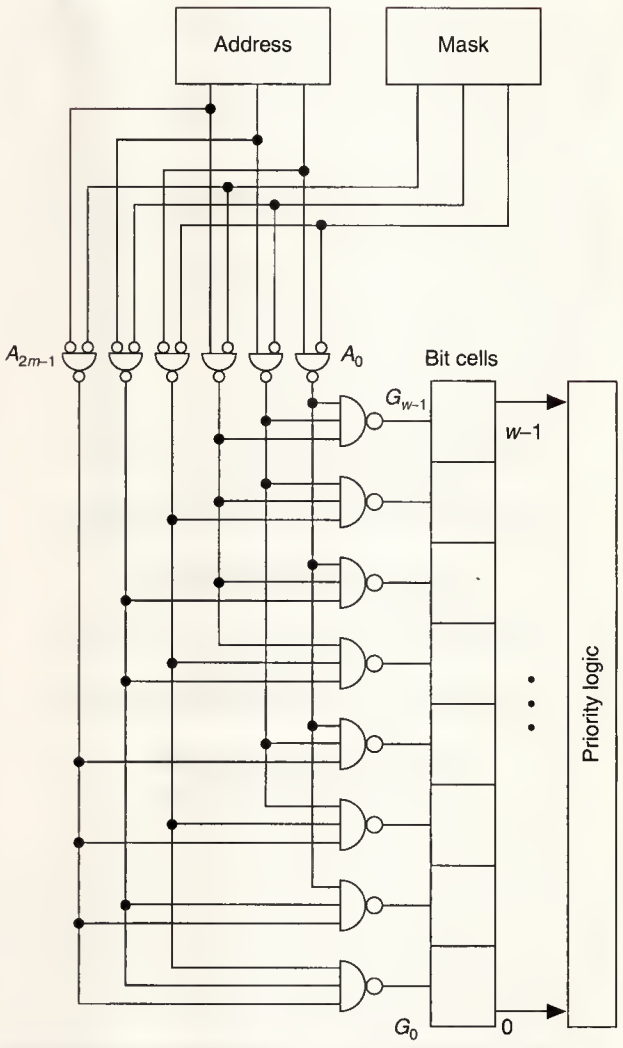


Figure 5. Architecture of the ARAM.[9]

additional counter field of 5 bits in each word implements extensions of this data word length. This field encodes 32 different positions in a larger extended data set. So the efficient word length can be varied between 32 bits, and 32×32 equals 1,024 bits.

The authors demonstrated several schemes using this CAM-oriented architecture for Prolog programming tasks such as variable binding, heap processing, and clause filtering. The system's host processor is a normal RISC-like processor. Several benchmark examples demonstrated the merits of the architecture by achieving very large processing speedups, while others showed smaller speedups.

Because CAMs have significant hardware costs, new approaches perform search operations directly at the interface between main memory and background mass memory media.

Ng, Clover, and Chung's approach is another example of such work.[16] The authors focus on hardware support for the binding of variables in Prolog program executions. In their approach, a CAM stores fact clauses as well as rule clauses. To maximize the speed of variable bindings, Ng, Clover, and Chung elaborated strategies to replace in parallel the variable contents of all matching expressions with their bound value. The system performs this directly when it detects a match—not only when it transfers variables to a special "bindings stack," as in former architectures for functional languages. The authors investigated different special CAM bit cells and developed a special 10-transistor CAM cell.

Naganuma et al. reported another associative approach for supporting Prolog execution.[17] Their processor consists of two subprocessors, one carrying out clause invocation, the other performing argument unification. Both subprocessors work in parallel. In this architectural scheme, CAM components mainly support a binding stack and a backtracking stack. These components are based on a CAM chip design of Ogura, Yamada, and Nikaido.[4]

Chae et al.[18] developed a CAM chip for a pattern inspection system. The chip comprises 256 twenty-five-bit words, together with shift registers and an address decoder/encoder. This storage memorizes 128 pattern words of 25 bits, each word with a corresponding 25-bit mask word. The size of the pattern words coincides with a 5×5 window in a binary pixel array; the corresponding mask word characterizes don't-care positions within this window.

Robinson's pattern-addressable memory (PAM) also uses single-instruction, multiple-data processor parallelism together with a memory that is content-addressable via hardwired pattern matching. The basic bit cell is a simple three-transistor DRAM cell, together with a comparator element formed by six transistors. Robinson presented a prototype PAM chip containing 1,152 twenty-bit words in 1989.[19] For a detailed description of the project's current status, see Robinson's contribution in this issue on pp. 20–30.

Another interesting approach for building processing logic around a CAM is the GLiTCH system,[20] used mainly to support vision processing. This chip contains 64 processing elements, each with 68 bits of local CAM. (Storer et al. present a detailed description of this architecture in another article of this issue, pp. 42–55.)

Mass memories and databases

CAM approaches have significant hardware costs. With the storage of large amounts of data, other essential limitations are

1) the memory must be loaded before it can be used for search operations, and
2) the fixed size of the array also necessitates a fixed comparison length and special efforts to change the format once it is selected.

Therefore, researchers have developed approaches to perform search operations directly at the interface between main memory and the slower background mass memory media (disk or tape). Such methods use the logic-per-track concept, which requires that comparison circuits be added to the read/write heads of disks or comparable media. The system checks for equivalence with search patterns on the fly when it transfers a stream of mass data to the main memory.

Using the logic-per-track concept, researchers have developed a number of "search engines." A major development was the Sure (Such-Rechner) system developed at the University of Braunschweig. Zeidler describes this system in a general survey of existing approaches.[21]

Lee and Lochovsky[22] describe Hytrem, a text-retrieval machine for large databases. It uses associative processing and a signature file that compresses the text pattern of a large database. Typically, this file takes up about 10 to 20 percent of the entire database. Hytrem has two major subsystems: a signature processor that compares a preprocessed search pattern against the signature file, and a pattern matcher that must eliminate false hits caused by considering only the compressed data.

Yamada et al.[23] developed another high-speed search machine. Besides processing logic, it uses a special 8-Kbit CAM, which holds 528 characters in a 16-bit character code. The CAM cell for storing one character consists of eight pair-bit CAM (PCAM) cells. Four conventional RAM bit cells and some hit logic make up a PCAM cell (see Figure 6). A PCAM cell can store one out of 10 different bit patterns, representing, for example, the four different combinations of two stored bits (0/0, 0/1, 1/0, 1/1), and also states like don't care or cleared.

Parhami recently proposed a serial/parallel architecture for a search machine.[24] This architecture consists of a linear array of processing elements, each element being connected to a circular shift register and a systolic array of comparator cells. Parhami shows how these elements cooperate to process a search in strings of data, and estimates the performance trade-off between serial and parallel search strategies. (Recently, Faudemay and Mhiri reported another approach for supporting string search operations with associative circuits.[25])

ARAM-based processors

From search operations, Tavangarian[26] generalized the flag-oriented ARAM concept to the processing of data. The operands of these generalized operations are flags. Initially, a number of w-bit data are compressed into one 2^w-bit-wide flag vector. A system using an extension of the ARAM structure (described earlier) can memorize several of these flag vectors and use appropriate Boolean functions to efficiently process them (often in parallel). Tavangarian describes a complete flag algebra of processing operations such as forming the union or intersection of flag vectors and checking for equivalence or antivalence between the vectors.

Another application of ARAM

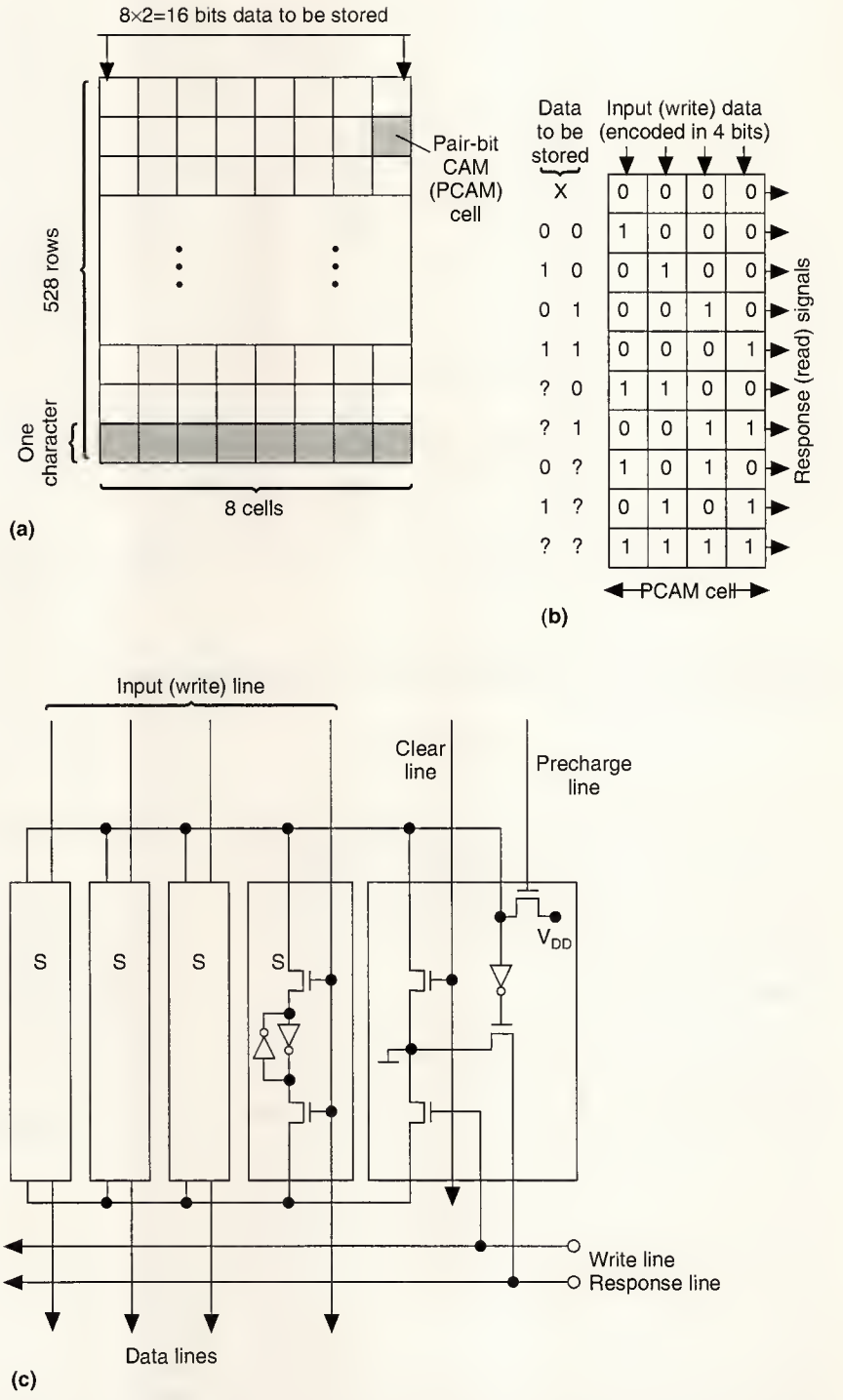


Figure 6. Pair-bit CAM cell: general bit cell array (a), coding of data (X indicates cleared, and ? indicates don't care) (b), and circuit structure (c). S indicates a SRAM bit cell.[23]

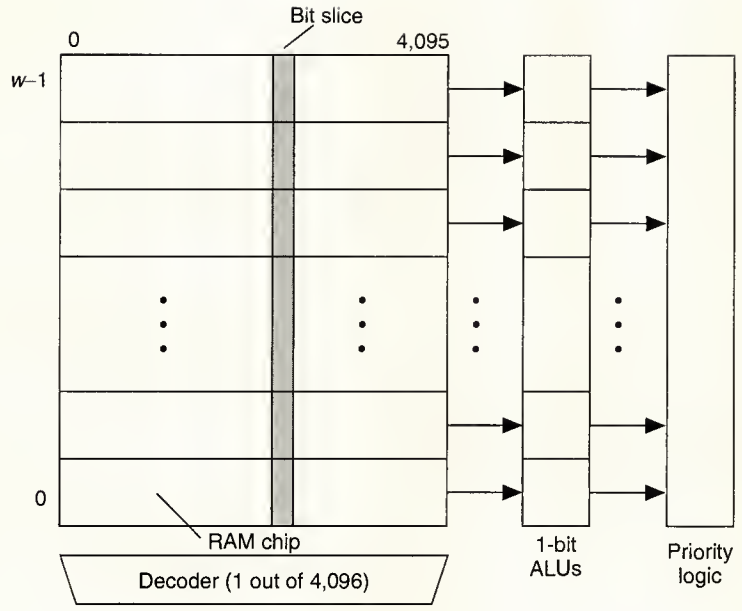


Figure 7. LUCAS architecture.[28]

components is their use in the associative processor AM[3] (associative multipurpose microprogrammable monoprocessor). This processor, based on AMD 2900 chips connected with associative memory, is intended mainly for applications like sensor control and speedup of CAD workstations.[27]

Parallel cellular logic

Some associative systems not only place CAM bit cells into a surrounding processing logic, as described earlier, but additionally integrate functionally complete arithmetic or Boolean processing elements into the individual word or bit cells.

A first approach toward this goal was the LUCAS (Lund University Content-Addressable System) architecture developed by Fernstrom, Kruzela, and Svensson.[28] LUCAS is a bit-sequential, word-parallel associative processor system. The CAM words are very long (4,096-bit word length), and a 1-bit arithmetic logic unit is associated with each word (see Figure 7). In hardware, one ordinary 4-Kbit RAM chip realizes each memory word. The system implements the entire search by transferring the memory content—bit slice by bit slice—to the ALU slice. The ALUs compare the bits of a bit slice in parallel with the bits of the given search pattern.

The advantage of this architecture is its easy implementation with already existing hardware building blocks. The tradeoff is the (often awkward and time-consuming) bit-slice-sequential organization of data processing and even simple retrieval operations. The authors also present a set of benchmark examples for evaluating the time complexity of basic application tasks for CAMs.

A recent promising approach for general-purpose associative systems is the associative string processor (ASP) developed by Lea at Brunel University.[29] The ASP system is a dynamically reconfigurable structure of communicating ASP substrings. Each substring contains a set of identical associative processing elements. Each element consists of an n-bit data register (n can vary, dependent on the application class, from 32 to 128 bits), an a-bit activity register (a varies from 4 to 8 bits), an $n + a$-bit parallel comparator, a single-bit full adder, and four status flags.

In another recent approach[30] I've tried to combine the ideas of Lea,[29] Tavangarian,[9] and Waldschmidt[10] and extend their solutions to achieve the following objectives:

- increase the flexibility of the logic elements, and
- combine processor cell arrays with ordinary CAM and RAM parts.

The main architectural features are

- extension, with relatively low hardware effort, of the usual comparison logic of a CAM cell to provide an entire set of 1-bit Boolean operations;
- extension of arithmetic elements from one sequential 1-bit adder element per word cell to at least 4-bit adder elements; and
- features for multiaccess to memory cells and ALUs.

A more detailed discussion of this architecture will appear in a future issue of *IEEE Micro*.

THIS ARTICLE SURVEYS THE GROWING number of recent research projects and implementations in the field of associative processors and memories. The increasing spectrum of approaches shows that we now have the potential for a renaissance of interest in these structures, especially as add-ons to existing architectures. Associative architectures can considerably improve the performance of today's computing systems in a growing spectrum of tasks; some of these potential applications will be discussed in more detail in the

subsequent articles in this issue. We might be seeing now—for the first time in the history of designing associative systems—a real chance to market them.

References

1. T. Kohonen, *Content-Addressable Memories*, Springer-Verlag, Berlin, 1980.
2. L. Chiswin and R.J. Duckworth, "Content-Addressable and Associative Memory: Alternatives to the Ubiquitous RAM," *Computer*, Vol. 22, No. 7, July 1989, pp. 51-64.
3. H. Kadota et al., "An 8-Kbit Content-Addressable and Reentrant Memory," *IEEE J. Solid-State Circuits*, Vol. 20, No. 5, Oct. 1985, pp. 951-956.
4. T. Ogura, S. Yamada, and T. Nikaido, "A 4-Kbit Associative Memory LSI," *IEEE J. Solid-State Circuits*, Vol. 20, No. 6, Dec. 1985, pp. 1277-1282.
5. T. Ogura et al., "A 20-Kbit Associative Memory LSI for Artificial Intelligence Machines," *IEEE J. Solid-State Circuits*, Vol. 24, No. 4, Aug. 1989, pp. 1014-1020.
6. J.P. Wade and C.C. Sodini, "Dynamic Cross-Coupled Bit-Line Content-Addressable Memory Cell for High-Density Arrays," *IEEE J. Solid-State Circuits*, Vol. 22, No. 1, Feb. 1987, pp. 119-121.
7. A. Kokubu, M. Kuroda, and T. Furuya, "Orthogonal Memory—A Step Toward Realization of Large Capacity Associative Memory," *Proc. Int'l Conf. VLSI*, North-Holland, Amsterdam, 1985, pp. 159-168.
8. H.E. Batcher, "STARAN Parallel Processor Hardware," *AFIPS Conf. Proc.*, Vol. 43, 1974, pp. 405-410.
9. D. Tavangarian, "Ortsadressierbarer Assoziativspeicher," *Elektronische Rechenanlagen*, (*Location-Addressable Associative Memory*), Vol. 25, No. 5, 1985, pp. 264-278.
10. K. Waldschmidt, "Associative Processors and Memories—Overview and Current Status," *Proc. Compeuro*, 1987, pp. 19-26.
11. C.C. Foster, *Content-Addressable Parallel Processors*, Van Nostrand Reinhold, New York, 1976.
12. K.J. Thurber and L. Wald, "Associative and Parallel Processors," *Computing Surveys*, Vol. 7, No. 4, Dec. 1975, pp. 215-255.
13. S.S. Yau and H.S. Fung, "Associative Processor Architecture—A Survey," *ACM Computing Surveys*, Vol. 9, No. 1, Mar. 1977, pp. 3-27.
14. J.G. Delgado-Frias and W.L. Moore, eds., *VLSI for Artificial Intelligence*, Kluwer Academic Publishers, Boston, 1989.
15. P. Kogge et al., "VLSI and Rule-Based Systems," in *VLSI for Artificial Intelligence*, J.G. Delgado-Frias and W.R. Moore, eds., Kluwer Academic Publishers, Boston, 1989, pp. 95-108.
16. Y. Ng et al., "Unify with Active Memory," in *VLSI for Artificial Intelligence*, Kluwer Academic Publishers, 1989, pp. 109-118.
17. I. Naganuma et al., "High-Speed CAM-Based Architecture for a Prolog Machine (ASCA)," *IEEE Trans. Computers*, Vol. 37, No. 11, Nov. 1988, pp. 1375-1384.
18. S.-I. Chae et al., "Content-Addressable Memory for VLSI Pattern Inspection," *IEEE J. Solid-State Circuits*, Vol. 23, No. 1, Feb. 1988, pp. 74-78.
19. I.N. Robinson, "The Pattern-Addressable Memory: Hardware for Associative Processing," in *VLSI for Artificial Intelligence*, Kluwer Academic Publishers, 1989, pp. 119-130.
20. A.W.G. Duller et al., "Design of an Associative Processor Array," *IEE Proc.*, Vol. 136, Part E, No. 5, Sept. 1989, pp. 374-382.
21. C. Zeidler, "Content-Addressable Mass Memories," *IEE Proc.*, Vol. 136, Part E, No. 5, Sept. 1989, pp. 351-356.
22. D.L. Lee and F.L. Lochovsky, "HYTREM—A Hybrid Text-Retrieval Machine for Large Data Bases," *IEEE Trans. Computers*, Vol. 39, No. 1, Jan. 1990, pp. 111-123.
23. H. Yamada et al., "A High-Speed String-Search Engine," *IEEE J. Solid-State Circuits*, Vol. 22, No. 5, Oct. 1987, pp. 829-834.
24. B. Parhami, "The Mixed Serial/Parallel Approach to VLSI String Processors," *IEE Proc.*, Vol. 136, Part E, No. 5, Sept. 1989, pp. 202-211.
25. P. Faudemay and M. Mhiri, "An Associative Accelerator for Large Databases," *IEEE Micro*, Vol. 11, No. 6, Dec. 1991, pp. 22-34.
26. D. Tavangarian, "Flag-Algebra: A New Concept for the Realization of Fully Parallel Associative Architectures," *IEE Proc.*, Vol. 136, Part E, No. 5, Sept. 1989, pp. 357-365.
27. M. Schulz et al., "An Associative Microprogrammable Bit-Slice-Processor for Sensor Control," *Proc. Compeuro*, Part 3, 1989, pp. 87-95.
28. C.F. Fernstrom, I. Kruzela, and B. Svensson, *LUCAS Associative Array Processor*, Springer-Verlag, Berlin, 1986.
29. M. Lea, "ASP: A Cost-Effective Parallel Microcomputer," *IEEE Micro*, Vol. 8, No. 5, Oct. 1988, pp. 10-29.
30. K.E. Grosspietsch, "An Architecture for an Intelligent Multi-Mode WSI Memory," in *Wafer Scale Integration III*, M. Sami and F. Distante, eds., North-Holland, Amsterdam, 1990, pp. 211-222.

The author's biography, picture, and address appear on p. 11 in this issue.

Pattern-Addressable Memory

Associative memories, such as those in caches and translation look-aside buffers, have proven their suitability for dealing with the dynamic and unpredictable aspects of runtime data. Pattern-addressable memory extends this capability from simple data words to the complex data structures generated and used by applications at runtime. The PAM chip is a custom associative memory specialized to handle the syntax and associated pattern-matching rules common to a range of symbolic processing applications. An array of these chips forms an associative coprocessor for a workstation.

Ian N. Robinson

Hewlett-Packard Laboratories

As applications are designed to interact more closely and intelligently with the real world, so the dynamics and uncertainty of their environments are reflected in the data structures they generate and use. Examples include the control of an automated factory floor with its myriad of interacting and failure-prone processes or a "simple" dialog with a user. Such interactive applications also necessitate timely responses, making rapid access to these runtime data structures important.

Associative memories have proven their suitability for fetching data that is dynamic, unordered, and unpredictable, as evidenced by the widespread use of associative hardware in cache memories and translation look-aside buffers. The associative coprocessor architecture described in this article provides rapid storage for, and access to, symbolic data structures generated at runtime.

The coprocessor's design is based on a custom VLSI associative memory chip. Two goals drove the chip design. The first was to provide hardware support for the syntax and pattern-matching rules used by symbolic expressions. The second was to overcome the storage density problem inherent in previous associative memory designs.

The resulting PAM chip stores a number of arbitrary length symbolic expressions in a format that requires little encoding overhead. The stored structures can be retrieved, modified, or deleted by pattern matching against an input structure (hence the name, pattern-addressable memory). The prototype coprocessor board, designed to work with a workstation host, uses an array of these PAM chips.

Declarative data structures and their access

Applications of particular interest are those in which runtime information is captured in the form of a database of declarative expressions. They are declarative because the control flow information (how, when, and by which process the expressions are to be accessed) that would otherwise allow them to be run procedurally is absent. Examples range from updating real-time database systems to handling the assertion and retraction of facts, constraints, and rules in knowledge-based systems. Such data structures are essentially interpreted at runtime, their activation being based on pattern matching against some other structure encoding a current event, query, or goal in the application. This access mechanism is central to the performance of the system. Unfortunately, pattern matching against a database of expressions can be a computationally expensive task.

To avoid an exhaustive linear search of the

0272-1732/92/0600-0020$03.00 © 1992 IEEE

database (or knowledge base), many designers have used indexing mechanisms, commonly based on hashing, to support access. Even indexing on the simplest structures, however, can be rendered impractical when the update rate is too high. Stormon gives the example of applying queries to the value entries of a stocks and commodities database.[1] Since this value is updated hundreds of times a second, it is not practical to maintain an index on it, and so answering such queries requires a sequential scan of the database records.

With the intricate indexing schemes necessary to efficiently access complex knowledge bases, these systems can tolerate significantly less dynamism. For example, the stored data often has a complex structure and arbitrary length, as opposed to the sets of uniformly formatted data types found in databases. Moreover, applications typically require very general access to the knowledge base, which in turn requires that indexes be maintained on all fields of the stored data. Indexing is also complicated by the use, particularly in stored structures, of wild cards or variables, which allow generalizations or partial knowledge to be stored.

These characteristics lead to such schemes as discrimination nets[2] in which trees of hash tables are constructed, each table associated with a particular combination of elements in the structure. Other methods for handling such associative lookup have also been built into the compilers for AI languages that use declarative programming, such as Rete networks for OPS5[3] and WAM (Warren Abstract Machine) code for Prolog.[4]

Information acquired after compile time presents a problem. As it interacts with its environment, an application can dynamically acquire, modify, and delete knowledge, including sensor information, user constraints, or changes in its control plan caused by external events. Even when the external world exerts little pressure, the internal dynamics can be considerable. Consider an application that uses hypothesis generation and test as a reasoning mechanism or reasons via constructing possible world scenarios. Such approaches involve adding new rules to the knowledge base at runtime and employing them to test their efficacy. These rules may then be modified or deleted and new ones generated in their place.

The problem, then, becomes one of maintaining efficient access to these transitory data structures. The compile-time techniques mentioned earlier can be adapted to run incrementally, but with the overhead of maintaining the indexing structures–an operation that impacts runtime performance. In the worst case, in which the indexing schemes are highly interdependent, adding new data could trigger a recompilation of the entire knowledge base. The consequences for the system can be more serious than merely slowing it down. In many cases there is a real-time constraint on the applicability of the information being accessed. For example, information on how best to avoid colliding with another moving object is of little use to the system after impact.

Symbols and expressions

Consider the example of a hypothetical application overseeing a robotized factory floor. An *X-Y* grid of locations divides the floor, and the robots must ferry parts about the factory. As part of its function, this application maintains a database of expressions denoting the location, cargo, and identification numbers of the robots.

The first expression below defines the format for these data structures. The second states that robot 2 is currently at location (3, 9) carrying some nuts and bolts.

```
( robot, ?loc, ?cargo, ?id )
( robot, ( at, 3, 9 ), [ nuts, bolts ], 2 )
```

As shown by this example, expressions can consist of constants (such as "at," "nuts," and "3"), variables (such as "?id"), and the parentheses that delimit substructure and lists. Moreover they can be of arbitrary complexity and length. To represent and manipulate expressions with a minimum of overhead, the PAM stores, matches, and outputs them as simple strings of symbols in an as-written order. Each symbol occupies one word of memory. Words are 32 bits wide, for compatibility with the host system, and are composed of a 4-bit type tag and a 28-bit name field. The name field can contain an integer (in the case of integer constants) or a symbol ID generated and understood by the application.

The PAM syntax represents these expressions as follows:

```
@robot ?loc ?cargo ?id
@robot ( at 3 9 ) ( nuts bolts ) 2
```

The header symbol (@) indicates to the hardware the start of a new expression and takes the place of the first level of parentheses. By convention the name part of the header stores what, in a database, would be the relation name or, in logic programming, the functor. All substructure, including lists, is uniformly represented (as far as the hardware is concerned) by opening and closing parentheses. Substructure can be arbitrarily nested.

An important part of a PAM symbol's definition is what it matches. Headers, for instance, only match headers and only then if the names are the same. Constants match other constants if the names match. Variables match other variables, constants, or entire substructures. Pattern matching ignores variable names. According to these pattern-matching rules, therefore, the two expressions above match each other.

PAM has three additional symbol types: list variable, trit word, and empty. Languages such as Lisp and Prolog use a special variable to represent an arbitrary number (including zero) of list elements, frequently at the tail of a list. This concept is carried through to the PAM as the list variable (represented

here by a preceding &) that can match any number of symbols within an expression. So, for example, a cargo list containing "bolts" can be sought using the following expression:

```
@robot ?loc ( &stuff bolts &stuff ) ?id
```

The trit word uses the hardware's capability to match with don't cares at the bit level. Trits are binary digits with a third don't-care state, typically represented by an X.[5] Two bits are required to encode each trit, and the name part of the symbol can carry a 12-trit word. Longer trits can be composed of consecutive words. The ability to store and match using trits allows numeric ranges and inequalities to be represented. For example, the expression

```
@robot ( at 10XX 11XX ) ?cargo ?id
```

will match any @robot expression describing a robot whose location falls within the square bounded by (8,12), (11,12), (11,15), and (8,15). Lastly, the empty symbol fills unused locations within the PAM. Nothing matches an empty.

PAM overview

Figure 1 shows a block diagram of the PAM board. It contains an array of custom-designed VLSI PAM chips (each containing symbol storage and processing logic) and an array controller. The latter communicates with the array via shared global data and instruction buses and with the host over the system's backplane. The PAM chips operate in parallel during instructions such as Match. The array controller's design lets multiple boards operate in parallel attached to the same host.

Each PAM chip in the array maintains its own symbol storage as a stack. The array controller selects which chip stores a particular expression. Expressions are input header first, as a sequence of symbols, over the global data bus. The incoming symbols are stored in consecutive words, or slots, in that chip's stack. Associated with every slot is a cell of a shift-register chain that runs the length of the stack. A bit in this shift register marks the slot to be written to. The bit acts as a top-of-stack, or write, pointer. It shifts to the next slot after each symbol is written.

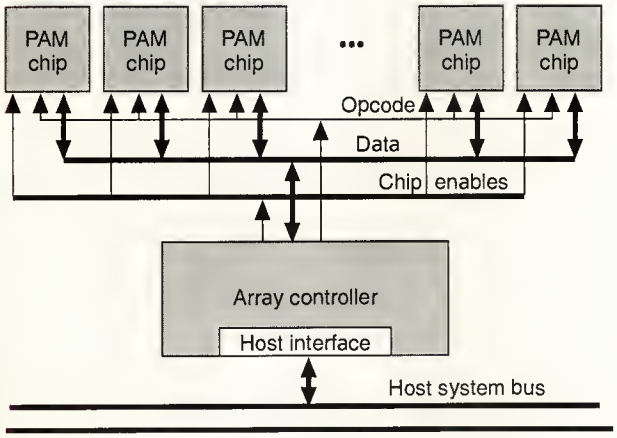


Figure 1. PAM board block diagram.

A second shift register with a corresponding read pointer connects slots back up to the global data bus for output. The outcome of the matching operation decides on which chip and where in its stack the read starts. The physical addressing of slots is not directly supported in the hardware; all access is mediated by associative pattern matching.

Matching. In pattern matching the controller broadcasts the query, header first, over the data bus. Every stored symbol on every PAM chip in the system matches itself against this sequence. Matching on the stored expressions therefore consists of a sequence of matches on their individual symbols. The states of each of these match sequences can be thought of as being represented by match tokens that move through the expressions as they match the incoming query. If a stored symbol matches the query symbol (according to the pattern-matching rules), and if the previous symbol is already flagged with a token, that token moves to the now matching slot. On a mismatch the token disappears. Headers are the only exception in that a header match ignores the previous match token state, essentially initializing the match sequence. Each slot incorporates storage for this match token state. At the end of query input the surviving match tokens mark the responders (expressions that pattern match).

The logic at each slot responsible for computing the new match token state is called the match engine. It is a combination of a slot-wide comparator and a finite-state machine, as shown in Figure 2. The comparator output and the match state of the previous slot are both inputs to the finite-state machine, its output being the new match state for that slot.

PAM supports two varieties of matching. The first is the conventional pattern match wherein variables can match constants or substructure, list variables can match any number of symbols, and so on. The second is exact matching, which requires the responder to match the query exactly, symbol for symbol. A user may prefer exact matching prior to deletion (described later) to select the more general form of an expression (such as "@robot ?loc ?cargo ?id") without selecting more particular forms ("@robot (at 3 9) ( ) 3").

Consider a pattern match between the query "@robot ?loc (& bolts &) ?id" and three stored expressions:

```
@robot ( at 3 9 ) ( nuts bolts ) 2
@robot ?loc ?cargo ?id
@robot ( at 3 9 ) ( ) 3
```

Note that the third expression has an empty cargo list and so should not match the query.

The first match is on the header symbols. This match creates tokens at all matching headers and only matching headers, regardless of the previous match state, as discussed earlier. After the query symbol has been applied, all the matching headers are marked. Asterisks mark the positions of match tokens.

```
query: @robot   stored:   @robot* ( at 3 9 ) ( nuts bolts ) 2
                          @robot* ?loc ?cargo ?id
                          @robot* ( at 3 9 ) ( ) 3
```

PAM can easily support a one-for-one match such as this, given the architecture in Figure 2. But what about the matches that are not one for one? For example, the next query symbol is a variable, and the match tokens in the first and third expressions enable substructure matches.

In both cases the match tokens have to be jumped simultaneously to the closing parentheses of those substructures. This must happen at the same time as a match occurs on the "?loc" in the second expression. A mechanism called the jump wire handles this procedure.

The jump wire carries tokens through the substructures in much the same way as a Manchester chain circuit propagates a carry signal through an ALU. Figure 3 illustrates the process. Initially the jump wire is precharged. A token marks the slot before an opening parenthesis. The query symbol "?loc" is then input. In the first phase, the match engines corresponding to headers and to slots carrying tokens cause the jump wire to be open-circuited (broken) at those points. In the second phase, match engines corresponding to opening parentheses with a preceding match token cause the discharge of their portions of the jump wire. In the third phase, match engines corresponding to closing parentheses that encounter a discharge in their portion of the jump wire set the match tokens in their respective slots.

Note that the parentheses alone carry no indication of nesting depth, so the jump marks all closing parentheses, as shown by the asterisks. An explicit nesting level could be added as the next symbol, but typically any spurious tokens will not survive through the match's duration. The resulting status looks like

```
query: ?loc   stored:   @robot (at 3 9 )* ( nuts bolts )* 2
                        @robot ?loc* ?cargo ?id
                        @robot ( at 2 9 )* ( )* 3
```

In the next step, as promised, the two extraneous tokens disappear. Now, however, the opposite situation occurs in which a stored variable "?cargo" matches substructure in the query. The variable enters the skip state (marked by a superscript S). This corresponds to holding on to the match token (not passing it on to the next match engine) while the matching substructure is skipped. While the engine is in this state, a token is released on each subsequent closing parenthesis that is entered.

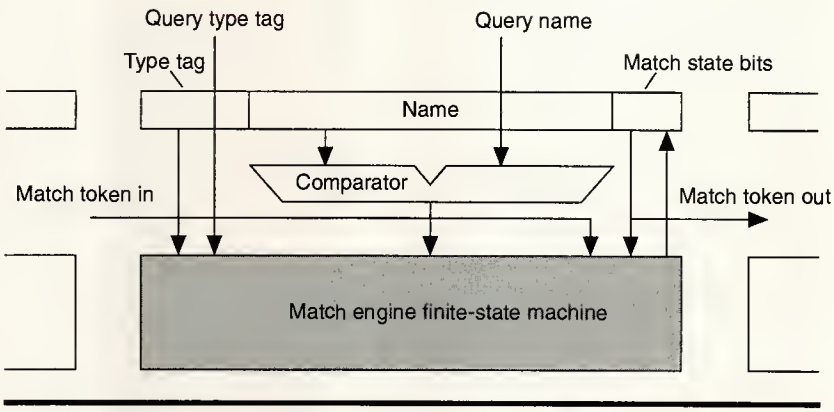


Figure 2. Slot and match engine.

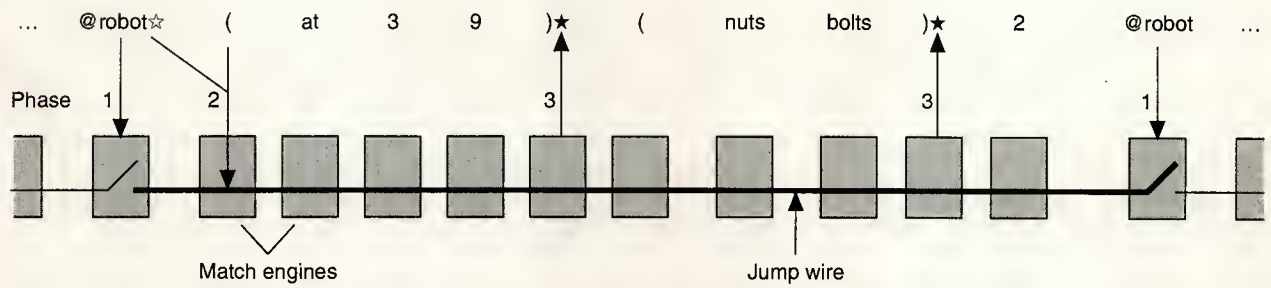


Figure 3. Jumping substructure.

query: (stored: @robot (at 3 9) (* nuts bolts) 2
@robot ?loc ?cargo^{S} ?id
@robot (at 2 9) (*) 3

A list variable in the query has an effect similar to, but more general than, an ordinary variable. Because the list variable is defined as matching any number of symbols, a jump marks every symbol in the remainder of the active expressions. In this way the next query symbol can continue the match at any point later in the expression. Any stored variables marked by the jump also enter the skip state, as the query could resume matching within a substructure matched by any of those variables.

query: & stored: @robot (at 3 9) (* nuts* bolts*)* 2*
@robot ?loc ?cargo^{S*} ?id^{S*}
@robot (at 2 9) (*)* 3*

In the opposite case of a list variable being activated in a stored expression, it is put into its own version of the skip state. In this state the stored list variable puts out a token on every query symbol entered.

No enabled symbols in the third expression match "bolts," and so all match tokens within that expression disappear.

query: bolts stored: @robot (at 3 9) (nuts bolts*) 2
@robot ?loc ?cargo^{S} ?id
@robot (at 2 9) ( ) 3

query: & stored: @robot (at 3 9) (nuts bolts*)* 2*
@robot ?loc ?cargo^{S*} ?id^{S*}
@robot (at 2 9) ( ) 3

Given a closing parenthesis all variables in the skip state put out a token.

query:) stored: @robot (at 3 9) (nuts bolts)* 2
@robot ?loc ?cargo^{S*} ?id^{S*}
@robot (at 2 9) ( ) 3

Finally a match on "?id" leaves match tokens at the ends of the responders.

query: ?id stored: @robot (at 3 9) (nuts bolts) 2*
@robot ?loc ?cargo^{S} ?id^{S*}
@robot (at 2 9) ( ) 3

Multiplexing and pages. In addition to input, output, and pattern matching, the PAM system also supports the *in situ* modification of stored expressions, their deletion, and garbage collection of the freed-up slots. The logic to support all these functions, described in more detail later, is rolled into that of the match engine. The ensuing complexity means that we must abandon the conventional scheme of physically attaching a match engine to each slot. Instead the PAM uses a word-serial multiplexing scheme to redress the balance between the areas occupied by memory and logic.

In the multiplexing scheme the memory stack on each chip is divided into a number of pages. The number of slots on a page equals the number of match engines. The match engines can then be collectively applied to each page in turn, as shown in Figure 4. Tokens or pointers leaving a page are latched in to the wraparound circuitry, which reintroduces them to the beginning of the match engine array on the next clock cycle, ready for the next page. This mechanism allows expressions to cross page boundaries.

During reading and writing only the page containing the slot of interest is active. During matching each query symbol input is held on the global data bus as the pages cycle through from the first to the last. Such an organization emulates the ideal situation of each slot having its own match logic. Figure 4 shows that this scheme is equivalent to associating a block of conventional RAM with each match engine. The page select mechanism becomes an address decoder shared by all the blocks. The array controller drives this page select bus, which is common to all chips in the array.

As a match sequence progresses, many pages will probably not contain any match tokens. Referring back to Figure 4, pages 1 and 3 would fit into this category. Within each chip, and across all chips in the array, a wire-Or of the new match token state is evaluated. The resulting signal to the controller can indicate a no match for that particular page. The page select logic within the controller cuts from the page sequence any pages that show a no match. This page-pruning scheme compensates in part for the performance penalty of multiplexing. In the best case, the page sequence will rapidly be cut down to the one page containing a responder. Matching then proceeds as if there were no multiplexing. The response can come even faster if there are no responders. The controller can signal this result as soon as all the pages are cut, often before the query is complete. Inactive pages are reactivated should a responder cross a page boundary onto one. This is done based on a signal from the wrap-around logic and, again, connected by a wire-Or between chips.

Another way to improve performance is for the application to exercise some control over data placement relative to pages. In this example, we may need only one page to store all the @robot expressions. If the controller knows this information a priori, it can restrict the query to only that page, resulting in ideal match times.

Dealing with responders. One action possible with responders is simply to output them. We can output the responders in their entirety or output only the currently marked slots. The latter is useful when the application is concerned more with the actions triggered by responders than with the responders themselves. The last slot of each stored expression

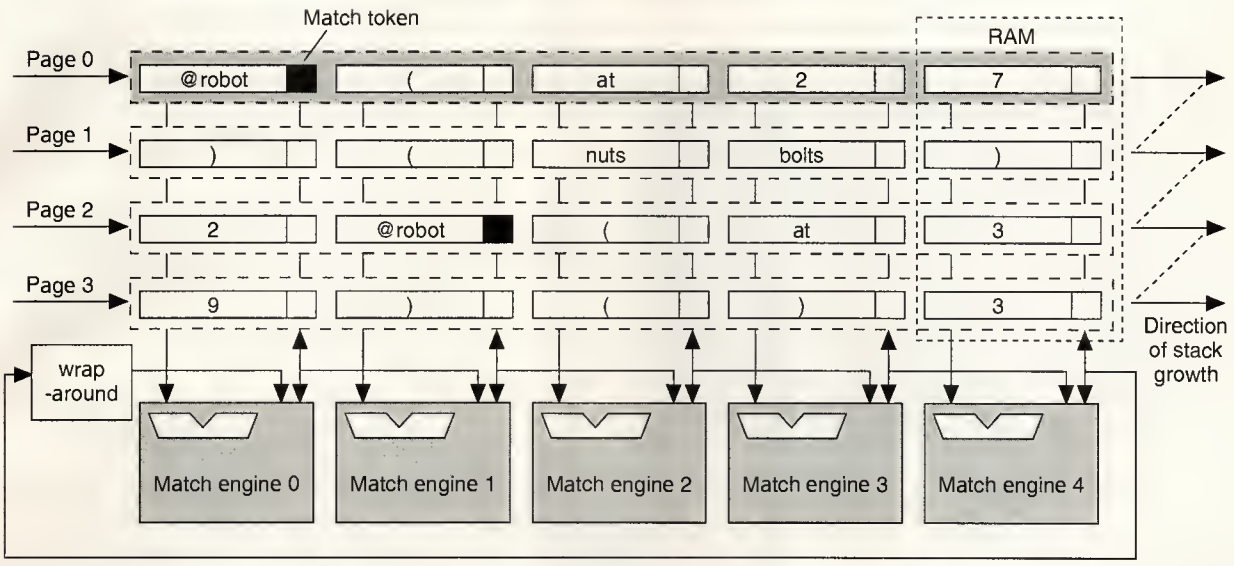


Figure 4. Pages and match engines (page 0 selected). This example shows multiplexing based on a hypothetical PAM chip containing five match engines and 20 words of storage (just enough to hold the two expressions shown). Tokens are shown after a match on @robot.

can store a pointer to code to be executed should that expression match. Such pointers can be matched by a final variable–for example, "?code_pointer"–in the query. This action will leave match tokens on the pointer slots of each responder.

Three major operations are connected with output: jump-read-to-token, retrieve-tokens, and read-slot. Jump-read-to-token, as the name suggests, uses the jump wire to move the read pointer on to the first or next slot marked with a match token. Jump-read-to-token is applied to each chip in turn. When no more match tokens are encountered on one chip, an overflow signal alerts the controller, which switches to the next chip. The chip select logic within the controller has pruning capabilities similar to those of the page select logic. The logic can remove chips (and entire boards in multiboard systems) from the scanning sequence if they contain no tokens.

The retrieve-tokens operation returns–in parallel, and again via the jump-wire–all the match tokens to the headers of their respective responders. In the jump, the wire breaks at the headers, match tokens discharge their associated wire segments, and new match tokens are created at headers that see a discharged segment below them. The jump-read-to-token operation will then move the read pointer to the beginning of each responder, rather than to its end. This facilitates the output of the entire responder. The read-slot operation simply enables the slot marked by the read printer onto the data bus. The read pointer also shifts so the next read-slot reads the next slot.

Modification. As an alternative to output, the PAM can directly modify responders. Doing so permits, for instance, the update of particular value slots accessed via a match on some attribute. Two mechanisms support this operation, and both rely on tokens marking the slots to be updated. The first mechanism updates slots individually and serially, using a read-modify-write operation. This operation can, for example, replace numerical values with their increments. The read pointer supports these updates as it moves to the relevant slots using the jump-read-to-token instruction.

To update all marked slots with the same value, the multiwrite operation makes full use of the PAM's parallelism. The symbol being written replaces the contents of all marked slots. Thus, for instance, all responders can be recorded within the PAM. If expressions are stored with an extra tag word at the end (similar–and, perhaps, in addition–to the code pointer described earlier), a new label can subsequently overwrite all the responders' tags. This label can be used to access them in later processing.

Deletion and garbage collection. Finally, the PAM can delete responders by marking all the slots within the responders and then multiwriting the empty symbol to them. It marks the slots by following a token retrieval with a match on a list variable.

Although removed from matching, the deleted expressions still occupy physical space in the chip's stack. Garbage collection can reclaim this space at any time. Garbage collection uses an alternating sequence of reads and writes, starting at

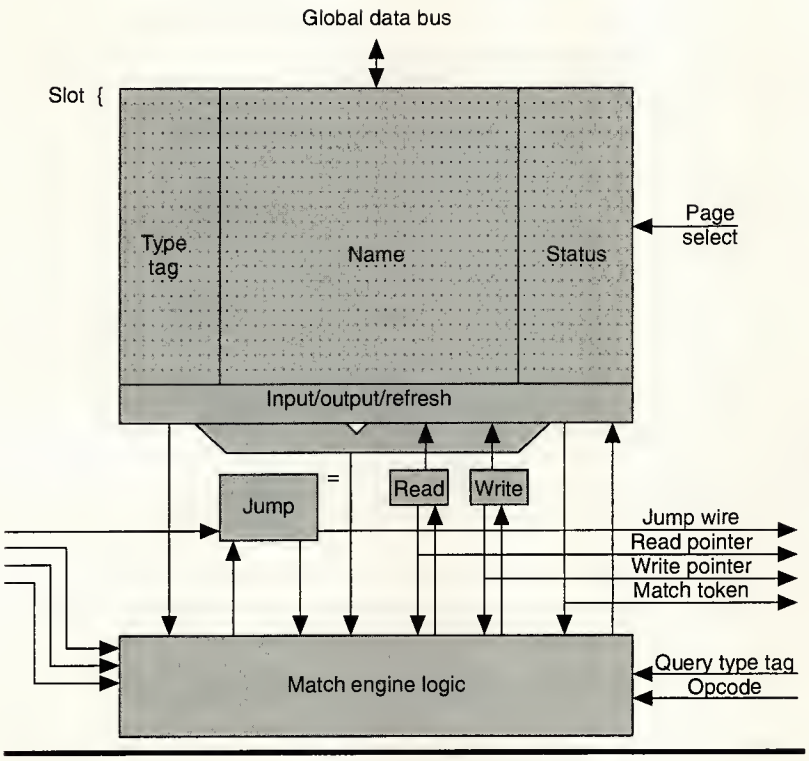


Figure 5. Block schematic.

the bottom of the stack, to essentially rewrite the contents without the empty slots. This is a serial process within each chip but is performed in parallel across all chips in the system. (A complete garbage collection in the prototype chips takes only 410 µs.)

PAM implementation

Some other associative memory designs support matching on symbolic data.[1,5-9] Such hardware has typically been based on traditional content-addressable memory (CAM), in which each memory cell contains a comparator circuit. This arrangement allows a match to be computed over all the stored words in parallel. Additional functionality, such as registers or simple ALUs, must be replicated for every word of storage. All this logic attached to each word limits actual chip capacities, despite the design of very dense CAM cells.[5,8] Also, the replication, plus the necessity of pitch matching with the CAM words, severely restricts the complexity of the additional logic.

In particular, none of the referenced designs can directly support pattern matching using the expression syntax (in both the query and the stored expressions) outlined in this article. Some aspects of the syntax can be encoded into a format suitable for comparators alone. For example, Kogge et al.[7] suggest a scheme in which a label is attached to each symbol denoting its position in a binary tree representation of the expression. This encoding, however, impacts both the runtime performance (similar to the software indexing schemes described earlier) and also eats into the space available for the expressions themselves. The alternative of adding to the complexity of the additional logic, with the frequency of its replication, quickly gives rise to unworkably poor memory densities.

The PAM's multiplexing scheme is similar in concept to that characterized by Yau and Fung[6] as block-oriented associative memory. At the time of that survey article, a block referred to the combination of a magnetic disk track, its associated read/write head (using a head-per-track scheme) and some match logic. The PAM's block comprises a small RAM and the match engine outlined earlier. The main advantage of this approach over other associative memory architectures is the memory density, and hence chip capacity, achievable by using a conventional RAM. Depending on the multiplexing ratio chosen, the overall storage density can approach that of the RAM alone. There are also advantages in implementation. One opportunity is to exploit the availability of off-the-shelf RAM block macro-cells from ASIC vendors.

The match-engine-plus-RAM block illustrated in Figure 5 forms the basic building block for laying out a chip. The layout of the block, particularly the multiplexing ratio, determines the overall memory density of the chip. A trade-off must be made, however, between capacity and processing speed. Extra multiplexing requires extra cycles to cover the extra pages. Using the page control strategies (described earlier) reduced this penalty.

A multiplex ratio of 16 was chosen for the prototype chip, partially because it yields an approximate 50-50 split between logic and RAM cell area. (Note that conventional RAMs are only about 60 percent memory cells. Sense amplifiers and addressing logic take up the rest of the area.) Since the system scans the memory repeatedly, it can use simple and dense dynamic RAM. The refresh of the slots' contents occurs along with the write-back of the new match token state.

The prototype chip uses a simple three-transistor DRAM cell. Figure 6 shows a bit column from the memory. Because the comparator uses both *n*- and *p*-type transistors in its discharge paths it does not require dual complementary inputs,

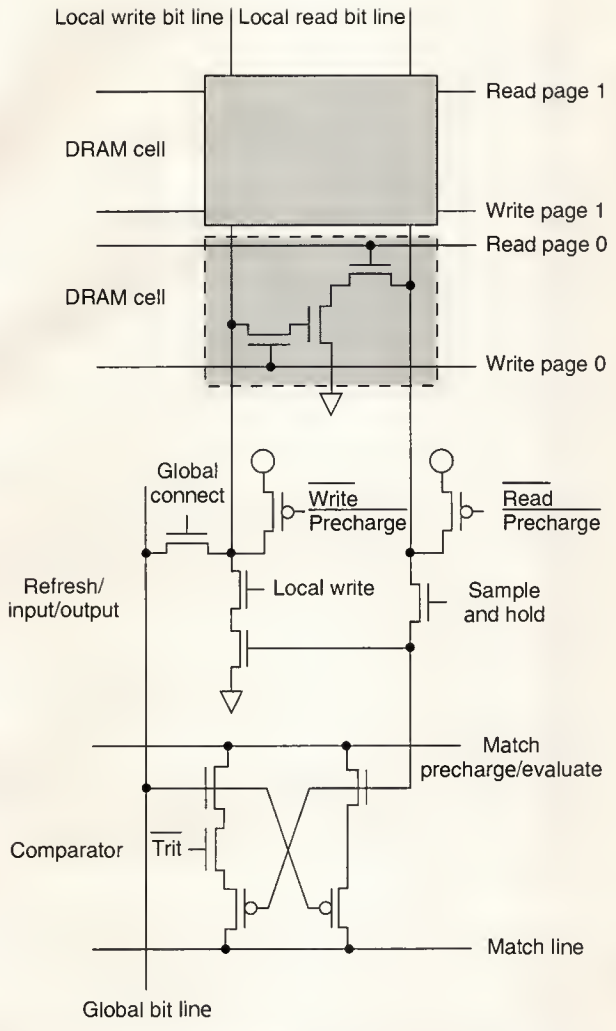


Figure 6. A bit slice showing one DRAM cell and its connections to the comparator and refresh/data I/O circuitry.

thus halving the wiring complexity. The extra gate on one side disables half of the comparator when trits are being compared. Trits are encoded using pairs of bits so that the two half-comparators can support the don't-care state (represented by the binary value 00 if stored or 11 if broadcast, in the case of the logic shown).

Because of its impact on the overall capacity of the chip, a great deal of effort went into minimizing the area occupied by the remainder of the match engine. Other than the circuitry implementing the jump wire and the read and write pointers, most of the match engine logic (shown in Figure 5) is compactly realized as a (much folded) PLA.

Using a 1.2-μm CMOS, two-level-metal process, an eight-by-eight array of these blocks resulted in a prototype chip containing 1,024 slots, each with 32 bits for the symbol plus 2 bits for the status, in an active area of 20 mm² (small by today's standards). The chip contains 64 match engines, and consequently there are 64 slots to a page. The match engine cycles in 200 ns (the design emphasized demonstrating functionality, rather than optimizing speed). In a cycle, the PAM reads the current page, evaluates the comparators and the jump wire, and subsequently writes back the new match token state.

Figure 7 is a photomicrograph of the prototype chip. The central vertical spine visible in the photograph carries the drivers for the page select, global inputs to the PLAs, comparators, and clocks.

Motomura et al. take a similar approach.[9] Their application concerns matching strings of characters. Although the syntax is not as wide ranging, the match engine is complex enough to make multiplexing worthwhile. They, too, use a 16-way multiplexing of the match engine logic. Instead of scanning pages, however, they use another on-chip CAM to index the correct page based on the first character input. The PAM could emulate this functionality by reserving page 0, say, for such an associative lookup table. Using a 0.8-μm, three-level-metal process, they integrated 160 Kbits of string-search memory onto a roughly 160-mm² die. With a similar technology and die size, the PAM's simpler architecture would yield a 1-Mbit capacity.

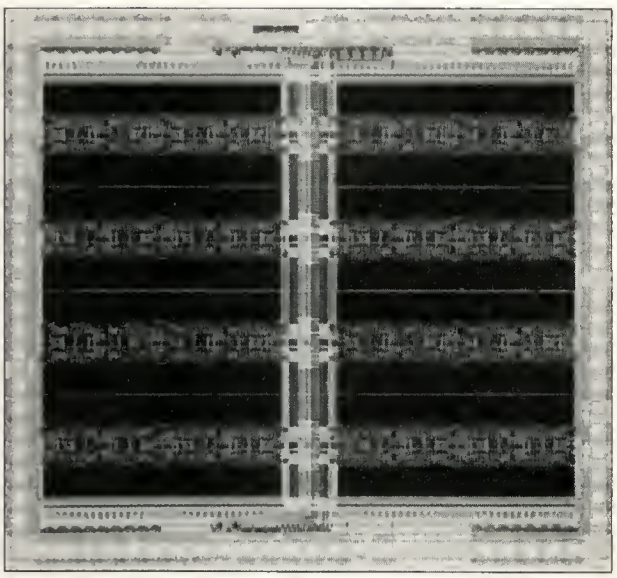


Figure 7. Prototype PAM chip.

Array controller. Figure 8 illustrates the three main parts of the array controller and its connections. The opcode generator handles instructions from the host and converts them into the correct sequence of PAM chip opcodes to perform that operation. It also coordinates the actions of multiple boards, should they exist. Table 1 summarizes some of the supported instructions. Other instructions provide more direct access to the internal state of the PAM chips and the controller, principally for debugging.

The page select logic selects the correct page for I/O operations as well as the page cycling and pruning functions. The chip select logic does much the same job at the chip level, selecting all chips for parallel operations (such as match) and individual ones for I/O. A similar mechanism to that used for pruning pages removes chips, and even entire boards, from this sequence if they contain no active tokens.

Figure 9 is a photograph of the completed prototype board residing in its host chassis, a Hewlett-Packard 9000 series 350 workstation. The board contains an array of 16 PAM chips. The array controller is implemented using the three large programmable logic devices near the top of the photo, to the left of the array.

Performance. Returning to the example of the robotized factory floor, assume that a central database records each robot's position and cargo. The database is stored in the PAM, where it is continually updated. The stored expressions are the same as in the pattern-match example. Note that the expressions are ordered so that the ID field is the last slot matched. Output can then access those IDs directly. Consider the following pair of queries:

- @robot (at 10XX 11XX) (& bolts &) ?id
  Are any robots within a unit or two of location (9.5, 13.5) carrying bolts?
- @robot (at 3 ?Y) ?cargo ?id
  Are any robots on the X=3 axis?

Note that both queries avoid indexing on the ID (such as, "Where is robot 4?"). Instead, we are accessing the more dynamic elements of the expressions. The @robots and the slot values within them are subject to constant insertion, deletion, and modification with little overhead besides reading or writing to the PAM's storage. Consequently, such updates have little effect on the match performance. Note that no garbage collection (even though it is fast) is required if only updates are allowed.

With the various page control schemes involved, times for the match and read-out functions will depend on the number of pages searched, and the number and distribution of partial and full responders through the chips and pages of PAM storage. Table 2 lists these results as averages of the best and worst case times for the various conditions, given one coprocessor board.

Using four pages, the PAM can track the status of 500 robots. Table 2 implies that such a system could support interleaved query and update rates of 85,000 expressions per second each. (Consider the sum of the average match, output, and update times.) This rate corresponds to allowing each robot's status to be updated roughly every 6 ms. Note that match times are relatively insensitive to the complexity or generality of the query.

The execution of Prolog programs can be thought of as an extension of this querying-a-database scheme. Prolog uses such pattern matching between subgoals and knowledge base clauses in its fundamental execution mechanism, unification. Unification is essentially pattern matching plus the handling of variable bindings. The latter is not a task particularly amenable to parallel hardware, however pattern matching alone provides an excellent initial filtering of candidate clauses for unification.[7,10]

The expressions stored in the PAM can also take on a more active role as triggers that are activated by particular queries. If these queries represent events observed by the system, then actions can be triggered by the responders to these events. For example, assume the robots of the previous example continuously report changes in their state to the PAM system as queries instead of updates. Some process at location (9.5, 13.5) that requires bolts could

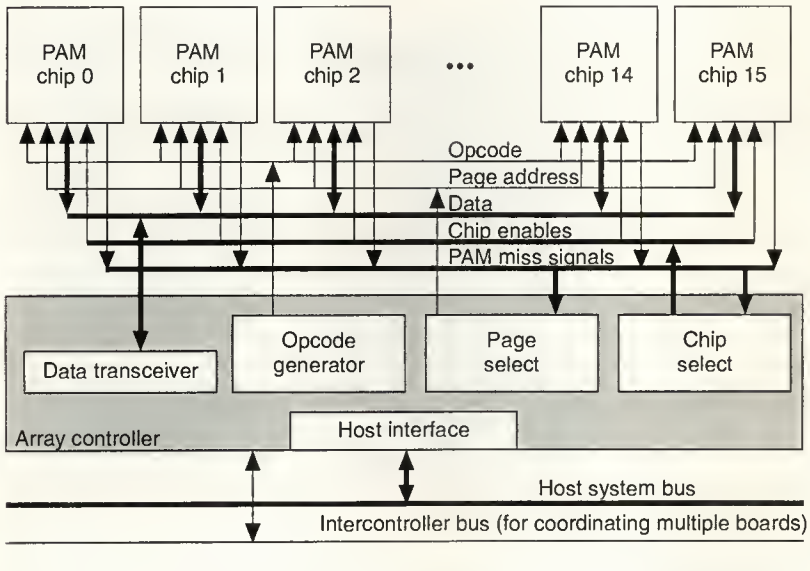


Figure 8. Coprocessor board schematic.

insert the following expression, and await a match.

@robot (at 10XX 11XX) (& bolts &) ?id

When a robot's status matches the expression, the absence of a no match signal (described earlier with the page-pruning scheme) signals the match. By appending a code pointer to the end of the stored expression, a routine could be activated to snag the passing robot. Just as before, the PAM can hold thousands of triggers and allow them to be constantly created, modified, and deleted. This operation style allows the interrupt-driven behavior popular in real-time control systems.

We can also think of triggers as templates that scan or parse the incoming expressions for sequences that pattern match. Since the same syntax is available to stored and query expressions, the expressions stored could be considered the query. We could use such an arrangement to scan the contents of a hard disk for matches. The PAM array's input bandwidth over its shared data bus is 32 bits every 200 ns, equivalent to 20 Mbytes/s. With current disk outputs of up to 5 Mbytes/s this rate sustains a multiplexing ratio of four, allowing the PAM to process hundreds of queries in parallel.

Triggers are also roughly equivalent to the situation action rules found in production system languages. Rules are fired based on particular combinations of events (or working memory elements). As with Prolog there is the added complication of handling variable bindings across condition elements. Kogge et al. demonstrate various ways of supporting this using associative hardware.[7] Although compilers exist for both Prolog[4] and OPS5[3] (one of the most popular production system languages), the PAM again provides the capability to handle dynamically created clauses and rules.

Blackboard systems[11] are a popular software architecture for AI systems applied to monitoring and control in complex environments. The blackboard provides a central knowledge base transparently shared by several knowledge sources. It establishes the context for knowledge processing actions, provides a repository for hypotheses, and controls the problem-solving process. Knowledge sources are scheduled based on events posted to the blackboard. These processes are associative in nature and commonly involve dynamic data. The PAM, therefore, also works well as a blackboard accelerator.

Lastly, in fields such as memory-based reasoning[12] and genetic algorithms,[13] systems attempt to reason or adapt themselves in the absence of rules. Such applications rely almost entirely on pattern matching and appear to be well suited to the capabilities of the PAM system.

THE PAM CHIP MEETS ITS DESIGN GOALS of supporting a rich expression syntax and pattern matching algorithm while achieving a high storage density. It accomplishes this by multiplexing complex processing logic over simple RAM. The use of RAM also makes implementation easier as no custom memory design is necessary. The prototype chips were fabricated and tested at speed. Current efforts focus on developing the supporting software, particularly with regard to having the board act as a full coprocessor, and compiling extensive in-system perfonmance data.

This same architecture can be adapted in a number of ways. The multiplexing ratio can be changed to yield different storage densities. In some applications the page management schemes described may allow significant increases in the proportion of RAM on the die. Also the match engine

Table 1. PAM instructions.

Instruction	Comments
write-slot <data> <chip>	–
write-all <data>	Write to all chips in parallel (saves time initializing the system)
pattern-match <data>	All match engines in parallel, page pruning enabled
exact-match <data>	(Same)
retrieve-tokens	All match engines in parallel
jump-read-to-token	Cycles through active pages, chips and boards looking for the next token
read-slot	–
read-modify-write <data>	–
multi-write <data>	All match engines in parallel
garbage-collect	All chips in parallel

Figure 9. Prototype PAM board in host system.

Table 2. Average match, output, and update times (in microseconds).

Number of responders	Match expression 1	Match expression 3	Output	Update
16 pages of @robots				
0	14.4	9.6	0.0	
1	15.1	10.2	0.6	16.5
2	15.3	10.5	1.2	
3	15.5	10.8	1.6	
4	15.7	11.1	2.0	
Four pages of @robots				
0	3.6	2.4	0.0	
1	4.3	3.0	0.6	5.7
2	4.5	3.3	1.2	
3	4.7	3.6	1.6	
4	4.9	3.9	2.0	

logic can be modified to handle, for instance, matching and approximate matching on character strings.[10]

The architecture enjoys all the bandwidth and scalability advantages of a logic-in-memory, single-instruction-multiple-data organization. The computational bandwidth on even the small prototype chip exceeds 1.2 Gbytes/s. The system can be scaled more or less arbitrarily, permitting more blocks to a chip (through a larger die and tighter design rules), more chips to a board (through better packaging), and more boards to a system. A system with four boards, each containing sixty-four 1-Mbit chips, would have a capacity of 32 Mbytes and an aggregate computational bandwidth of 1.2 terabytes per second.

Acknowledgments

This work originated as part of a computer design effort led by Al Davis, and I am indebted to him for his support and advice. Thanks also go to K.E. Grosspietsch for his help in preparing the article.

References

1. C. Stormon, "The Coherent Processor–A Content-Addressable Memory for AI and Databases," *Proc. Wescon*, Los Angeles, Calif., 1989, pp. 240-244.
2. E. Charniak, C.K. Riesbeck, and D.V. McDermott, *Artificial Intelligence Programming*, Lawrence Erlbaum Assoc., Hillsdale, N.J., 1980.
3. C.L. Forgy, "RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligence*, Vol. 19, 1982, pp. 17-37.
4. D.H.D. Warren, "Implementing Prolog," Tech. Report 39, Edinburgh University, UK, 1977.
5. J.P. Wade and C.G. Sodini, "A Ternary Content-Addressable Search Engine," *IEEE J. Solid-State Circuits*, Vol. 24, No. 4, 1989, pp. 1003-1013.
6. S.S. Yau and H.S. Fung, "Associative Processor Architecture–A Survey," *Computing Surveys*, Vol. 9, No. 1, Mar. 1977, pp. 3-28.
7. P. Kogge et al., "VLSI and Rule-Based Systems," *VLSI for Artificial Intelligence*, J.G. Delgado-Frias and W.R. Moore, eds., Kluwer Academic Publishers, Hingham, Mass., 1989, pp. 95-108.
8. T. Ogura, S. Yamada, and J. Yamada, "A 20kb CMOS Associative Memory LSI for Artificial Intelligence Machines," in *Proc. IEEE Int'l Conf. Computer Design*, CS Press, Los Alamitos, Calif., 1986, pp. 574-577.
9. M. Motomura et al., "A 1.2-Million Transistor, 33-MHz, 20-b Dictionary Search Processor (DISP) ULSI with a 160-kb CAM," *IEEE J. Solid-State Circuits*, Vol. 25, No. 5, Oct. 1990, pp. 1158-1165.
10. I.N. Robinson, "A Prolog Processor Based on a Pattern Matching Memory Device," *Proc. Third Int'l Conf. Logic Programming*, E. Shapiro, ed., Springer-Verlag, New York, 1986, pp. 172-179.
11. B. Hayes-Roth, "A Blackboard Architecture for Control," *J. Artificial Intelligence*, Vol. 26, 1985, pp. 251-321.
12. C. Stanfill and D. Waltz, "Toward Memory-Based Reasoning," *Comm. ACM*, Vol. 29, No. 12, Dec. 1986, pp. 1213-1228.
13. D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Mass., 1989.

Ian Robinson is a member of the technical staff at Hewlett-Packard Laboratories. His research interests include VLSI design, parallel computer architecture, and artificial intelligence.

Robinson received a BSc degree in physics with electronics from the University of Sussex. He is a member of the IEEE Computer Society.

Direct questions about this article to the author at Hewlett-Packard Laboratories, 1501 Page Mill Road, Building 3L, Palo Alto, CA 94304; or via e-mail at irobinson@hplinr.hpl.hp.com.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 159 Medium 160 High 161

A Dynamic Associative Processor for Machine Vision Applications

Massively parallel associative processors may be well suited as coprocessors for accelerating machine vision applications. They achieve very fine granularity, as every word of memory functions as a simple processing element. A dense, dynamic, content-addressable memory cell supports fully parallel operation, and pitch-matched word logic improves arithmetic performance with minimal area cost. An asynchronous reconfigurable mesh network handles interprocessor communication and image input/output, and an area-efficient pass-transistor circuit counts and prioritizes responders.

Frederick P. Herrmann

Charles G. Sodini

Massachusetts Institute of Technology

Recent technological advances have led to two developments with exciting implications for machine vision. First, massively parallel supercomputers have come of age. With many thousands of processing elements and some Gbits of total memory, these systems may be the most promising technology for high-level, image-understanding applications. Second, designers are applying VLSI to low-level, or *early*, vision problems. High density and low power make single-chip solutions feasible, perhaps on the same chip as the imager.

Somewhere between the massively parallel supercomputer and the application-specific analog solution a need exists for a simple, low-cost, very fine-grained machine. There is a class of applications in early and middle vision for which million-dollar supercomputers are overkill, yet analog solutions are insufficiently general. The associative parallel processor can fill this niche.

In the early years of associative processing, the pioneers of the field recognized picture processing as a promising source of applications for their new machines.[1,2] Image processing problems can be rich in inherent parallelism, with many thousands of pixels receiving identical processing steps. The low precision of image data (typically 8-bit integers) and the often modest computational requirements at each pixel match the limitations of bit-serial arithmetic. Associative processors are an attractive solution, because they are by nature fine-grained machines in which every word of memory functions as a tiny processing element (PE). Modern VLSI technology provides the density necessary to produce large arrays at low cost, with each PE assigned to a single pixel.

As shown in Figure 1 on the next page a 2D network connects the PEs and handles image input and output. A host computer broadcasts instructions to all PEs in the array. Two data conversion steps are needed to load the array with a digitized image. First, the imager makes an analog electronic signal representing the image. The acquisition domain is typically optical, but the imager could be an electron microscope, a medical magnetic resonance imager, or a radio telescope.

After any analog-domain processing, an analog-to-digital converter produces a digital signal for the associative processor. The edge of the network provides a high-bandwidth port for image input and output. This system can serve as the front end of a hierarchical architecture for image understanding[3] or as a stand-alone processor for pattern recognition and image processing tasks.

0272-1732/92/0600-0031$03.00 © 1992 IEEE

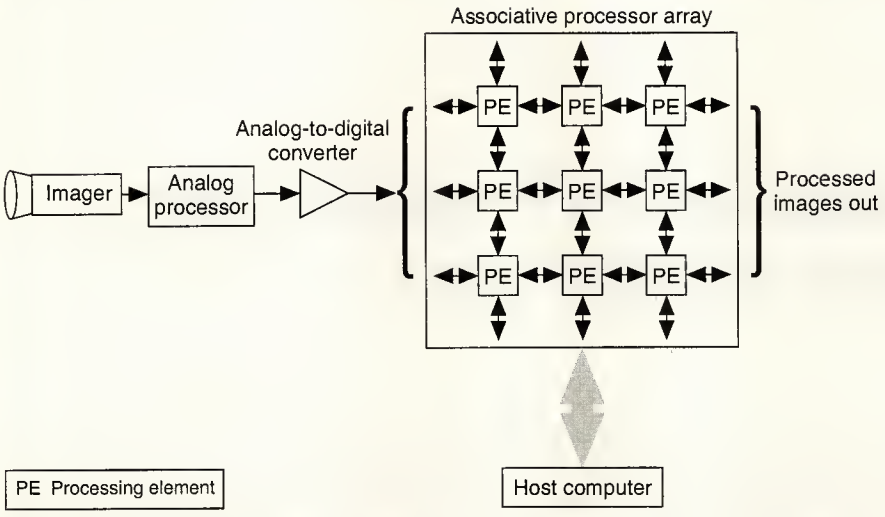


Figure 1. Vision system incorporating an associative processor.

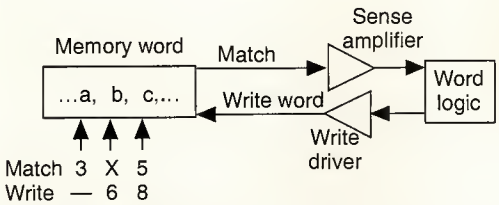


Figure 2. Associative PE.

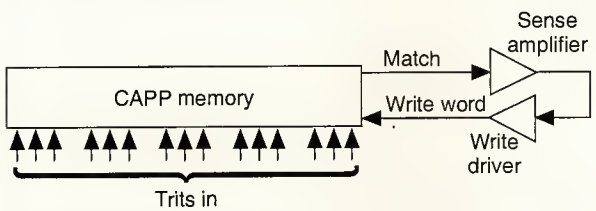


Figure 3. PE with direct match-write feedback.

Figure 2 is a generalized diagram of a single associative PE. The memory word stores patterns, which may have several fields ($a,b,c,...$). A match operation finds PEs that match one or more fields; unused fields are masked with don't cares (X). For example, the operation Match (3, X, 5) would identify all PEs with $a = 3$ and $c = 5$, and would set their sense amplifier outputs to 1. Each PE's match result passes to its word logic, which in turn controls the write driver. Thus, match results condition write operations. If the write driver is not enabled, the PE is masked, and its memory remains unchanged.

Fields within a word can also be masked, so we can modify some fields while preserving others. For example, Write (–, 6, 8) will set $b = 6$ and $c = 8$ while leaving the contents of the a field unchanged.

Associative parallel processors may be bit serial/word parallel or fully parallel. In the first case, match and write operations may examine or modify only one bit of each memory word at a time. Multibit operations take several cycles, and combining results of the single-bit matches requires relatively complex word logic. Fully parallel systems perform multibit matches and writes as single operations and therefore may use simpler word logic. The advantage of the bit-serial approach is that designers can use conventional RAM cells, while fully parallel systems require special content-addressable parallel processor memory.

This CAPP memory has historically been relatively expensive compared to standard RAM. However, a new dynamic cell[4] uses only five N-channel transistors and with equivalent technology should achieve a density similar to that of a CMOS SRAM. This new technology should make fully parallel systems competitive, especially in applications requiring many relatively simple PEs.

The dynamic CAPP cell (see box) stores the three ternary digits (trits) 0, 1, and X. The match operation compares the stored trit to a presented datum, with every trit matching itself, and the don't care (X) matching every trit. All the cells of a word perform the match comparison in parallel, and the word match result is the logical And of the individual cell match results.

The write operation modifies the cell contents. If a masked Write (–) is presented, the cell contents will be preserved. Otherwise the cell will take the value of the presented trit. Of course, none of the cells in the word will be modified unless the word logic activates the write enable driver.

The cell also supports a read operation, but the associative processor system does not use it. Instead, the processor array outputs data through the interprocessor communication network.

Word logic

In an area-efficient design the word logic must match the vertical pitch of the memory cells. This constraint is a strong incentive to reduce word logic complexity. Consider, as a design exercise, the simplest logic element imaginable: a wire.

Figure 3 is a fully parallel associative PE with the sense

Dynamic content-addressable parallel processor cell

Figure A is a circuit diagram of the dynamic CAPP cell used in the fully parallel associative processor. The cell uses five N-channel MOS transistors, including two overlapping dual-gate structures available in MIT's CCD/CMOS process.

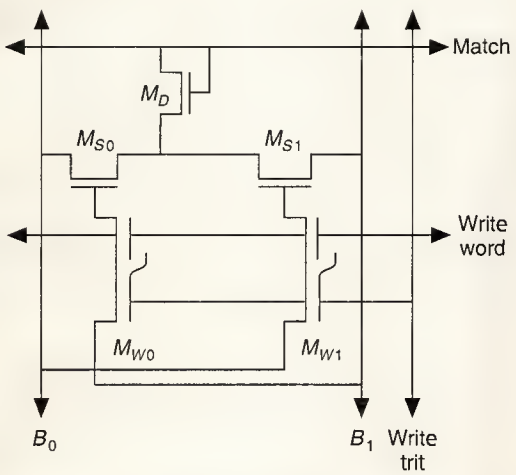


Figure A. Dynamic content-addressable parallel processor cell.

Charge is stored on the gates of M_{S0} and M_{S1}, which are written through the M_W devices. The diode-connected transistor prevents shorting the bit lines ($B_{0,1}$) of adjacent cells through the match line M. Three states (trits) may be stored in the cell: 0 or 1 by charging the gates of M_{S0} or M_{S1}, and X (don't care) by discharging both (see Table A). Because it is difficult to charge both gates simultaneously, no fourth state is used.

The cell perfonms match, read, and write operations. The match operation begins with the match line precharged to a high potential. The bit lines are then driven with the match trit (Table B), and a mismatch will cause the match line to be discharged. For example, a 1 is presented by dropping B_0 while B_1 remains high. If the cell is storing a 0, then M_{S0} will be on, and current will flow through M_D and M_{S0}. Similarly, a 0 is presented by dropping only B_1. If both bit lines remain high, then an X has been presented, as there can be no discharge path and a match is guaranteed. Finally, if both bit lines are driven low, the cell will indicate a mismatch if either storage device is on. This fourth (can't match) datum is denoted with the symbol Ø. Its primary use is to detect stored X's during refresh.

The read operation is similar to the match, except that the match line is used to pull up the bit lines, instead of the bit lines discharging the match line. If a 0 is stored in the cell, B_0 will be pulled up through M_D and M_{S0}. A stored 1 will cause B_1 to rise. If the cell is in the X state, both storage devices will be off, and both bit lines will remain low.

Two write lines are necessary to provide write enables in two dimensions. The word logic controls the write-word line, which runs horizontally. The write-trit line runs vertically and is used for trit-column masking. The cell is written by raising both write lines and driving the bit lines to the appropriate potentials. If the write-trit line is held low, the write is masked, and the state of the cell will remain unchanged. The symbol "–" denotes this masked write.

Table A. Three states stored in the cell.

State	Storage nodes	
	$V_{GS}(M_{S0})$	$V_{GS}(M_{S1})$
0	High	Low
1	Low	High
X	Low	Low

Table B. Control of match and write operations.

Operation	Control lines		
	B_0	B_1	Write trit
Match 0	High	Low	Low
Match 1	Low	High	Low
Match X	High	High	Low
Match Ø	Low	Low	Low
Write 0	Low	High	High
Write 1	High	Low	High
Write X	Low	Low	High
Write –			Low

amplifier output fed directly back to the write enable driver. Match and write patterns are presented to the CAPP memory word on the trit lines, which run vertically through the PE array. Surprisingly, even this simple PE can perform useful operations. We use a sequential state transformation process.[1]

Table 1 is a truth table for the destructive single-bit full add,

$A + B + C \rightarrow A, C.$

Table 1. Full-add truth table and transformations.

	Previous state			New state		
State	A	B	C	A	C	Transform
0	0	0	0	0	0	√
1	0	0	1	1	0	
2	0	1	0	1	0	
3	0	1	1	0	1	√
4	1	0	0	1	0	√
5	1	0	1	0	1	
6	1	1	0	0	1	
7	1	1	1	1	1	√

Table 2. Full-add procedure.

		Pattern		
Step	Instruction	A	B	C
1	Match	0	0	1
2	Write	1	–	0
3	Match	1	0	1
4	Write	0	–	1
5	Match	1	1	0
6	Write	0	–	1
7	Match	0	1	0
8	Write	1	–	0

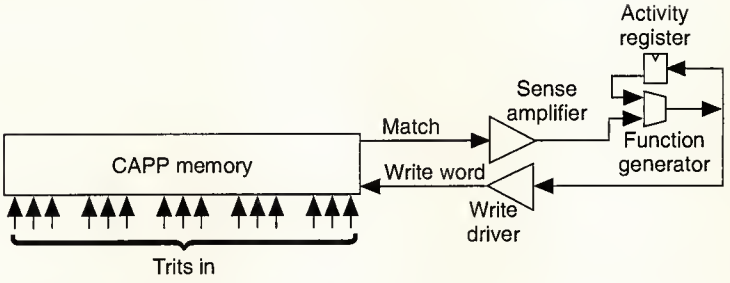


Figure 4. PE with improved word logic.

The least-significant bit of the sum replaces the value of A, and the carry bit C receives the most-significant bit. A quick examination of the table reveals that no work needs to be done in the checked states; the new values of A and C are the same as the old values. The other four states need to be transformed, as indicated by the arrows. Each transformation will require one match and one write operation, as shown in Table 2.

The first match operation selects all PEs in the 001 state. The Write (1 – 0) modifies the A and C bits appropriately, transforming state 1 into state 4. Steps 3 and 4 transform state 5 into state 1, and subsequent operations handle the remaining transformations. The procedure requires eight operations, but some are redundant: The four write instructions use only two write patterns. Perhaps a different choice of word logic could reduce the total number of operations.

Figure 4 shows the word logic actually used in this associative processor design. It includes a two-input function generator and one bit of state, the activity register (AR). The function generator can compute any of the 16 binary Boolean functions of two inputs. We define two operations:

- *Match.* The function generator is evaluated using the old values of the AR and sense amplifier (SA). The AR takes the function generator output as its new value, and the SA value is replaced by the match result on the CAPP memory word.
- *Write.* After evaluation, the function generator is used to enable the write driver. The AR and SA values remain unchanged.

Table 3 shows an add procedure for the improved word logic. In the first step, the processor performs a match with the pattern XX1. This loads C into the sense amplifier. In step 2, the function generator passes the sense amplifier result to the activity register, while the sense amplifier takes on the value B. The exclusive-Or ($B \oplus C$) is computed in the third step, while the value of A is loaded into the sense amplifier.

After three matches, the SA and AR contain the information necessary to enable the writes. Step 4 uses the function SA′ ∧ AR to enable the instruction Write (1 – 0), transforming states 1 and 2 of Table 1. The last write transforms states 5 and 6.

The function generator and activity register reduce the number of instructions required for a full add from eight to five. Table 4 presents results for other arithmetic operations and shows that the improved word logic can increase performance by a factor of two or more. (These figures represent per-bit requirements, excluding constant-time initialization or cleanup instructions.)

We must weigh the performance benefits provided by the word logic against its cost in silicon

Table 3. Add procedure for the improved word logic.

Step	Instruction	Function	Pattern A	Pattern B	Pattern C	Sense amplifier contents	Activity register contents
1	Match		X	X	1	C	
2	Match	(SA)	X	1	X	B	C
3	Match	(SA ⊕ AR)	1	X	X	A	$B \oplus C$
4	Write	(SA′ ∧ AR)	1	–	0	A	$B \oplus C$
5	Write	(SA ∧ AR)	0	–	1	A	$B \oplus C$

area. Fortunately, this cost is small. The activity register is almost cost free, since it shares many transistors with a shift register, which is required for testing. The function generator occupies about 5,000 square microns in 2-μm design rules or a little more than four memory cells. In the experimental chip discussed later, the function generators accounted for less than 5 percent of array area.

One might consider further increasing word logic complexity to provide greater performance. For example, a three-input functional unit, such as a full adder, might replace the two-input function generator. Doing so would trim the destructive add algorithm modestly, from five operations down to four, but word logic area would more than double. We deemed the two-input function generator a more appropriate trade-off between performance and area, consistent with our stated goal of very fine granularity.

Table 4. Cycles required for arithmetic operations.

Operation	Notation	State transformation	Improved word logic
Destructive add	$A + B + C \rightarrow A, C$	8	5
Nondestructive add	$A + B + C \rightarrow \Sigma, C$	12	5
Scalar add	$A + s \rightarrow A$	4	3
Absolute value	$\lvert A \rvert \rightarrow A$	6	3

Experimental implementation

We built an associative processing test chip in MIT's 2-μm CCD/CMOS process[5] as a first step toward complete implementation. Along with other key circuits, the chip includes an array of 64 PEs, each with 64 trits of CAPP memory (see Figure 5). We chose the number of trits per PE after considering the memory needs of several low-level vision algorithms. To ease the pitch constraint, however, the memory is actually implemented with two 32-trit words for each PE. In this way, two word pitches are available for laying out the sense amplifier, function generator, and activity register. A match line multiplexer allows these circuits to be shared by the two words, however each PE requires two write drivers and two match drivers.

Our prototype design relies on off-chip control logic, timing, and instruction decoding. Plans for the final implementation include these subsystems on chip, along with interprocessor communication and response resolution circuits. The anticipated density is 256 PEs per chip in 2-μm technology.

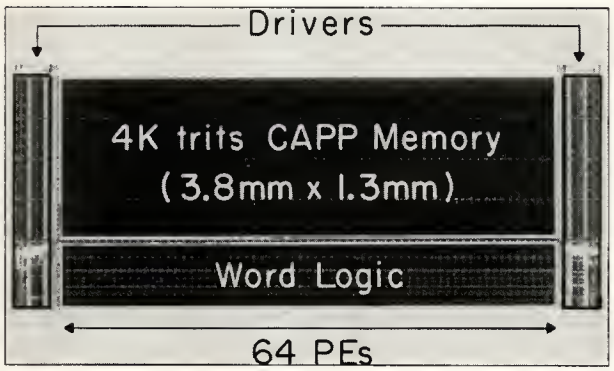


Figure 5. Detail from photomicrograph of experimental chip, showing array of 64 PEs with 4,096 trits total memory.

Interconnection

Interprocessor communication is an important consideration in the design of any parallel processor, and this associative processor is no exception. However, the associative processor's fine grain creates a different set of constraints than that which might limit a coarse-grained design. For example, an often-repeated maxim says that in modern VLSI systems, "Wires are expensive, and transistors are cheap." In a pitch-matched lay-

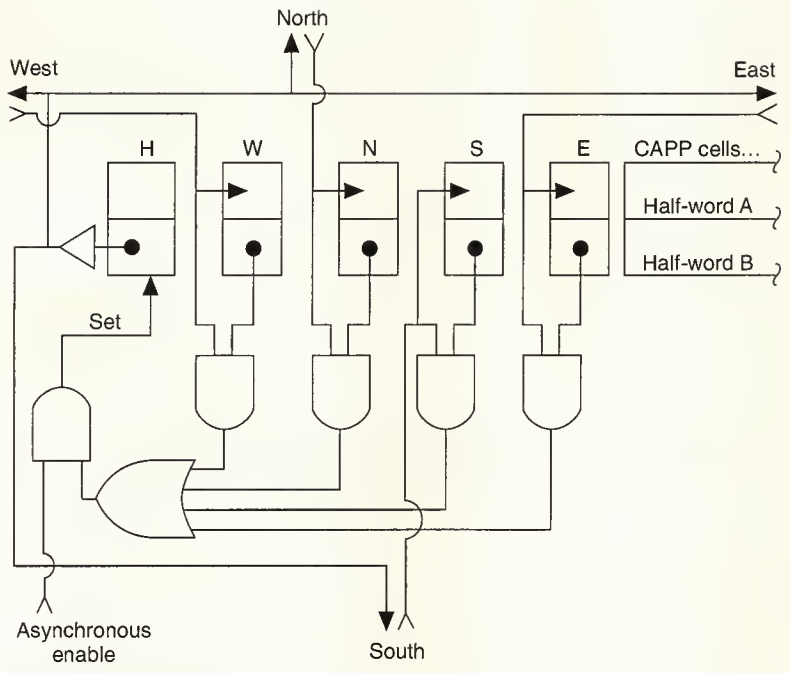


Figure 6. Special cells for network.

out, however, transistors are expensive too. Output circuits in particular demand considerable area. If a wire runs halfway across the chip, then the transistors must be large enough to drive several picofarads.

Efforts to reduce network wiring may result in false economies if they require additional drivers or if they require additional control signals to be routed through the array. Extensive multiplexing is more appropriate for interchip communication than for on-chip networks in very fine-grain systems.

Fortunately, most early vision algorithms have modest communication requirements. A 2D rectangular mesh topology provides a natural processor-to-pixel mapping, and the large diameter of the network is not a limitation when most communication is restricted to local neighborhoods. Unlike higher order networks, the mesh readily extends across chip boundaries at present and anticipated levels of integration.

How can we add network capabilities to the simple PE of Figure 4? The most common model treats the network as an extension of the word logic, adding special-purpose communication registers. Since the present word logic has only one register, this would considerably increase its complexity. Instead, our design preserves the word logic and extends the CAPP memory word with special network cells.

Figure 6 shows the 10 special cells for networking. Since the CAPP memory is already organized into two 32-trit words, we grouped the 10 cells into five cell pairs. The B half of each pair may be written and matched like a standard CAPP cell, except that it stores only two states. The A halves are simple match-only cells; network functionality does not require them to be writable.

There are four cell pairs–N,E,W, and S (one for each compass direction)–and a fifth Home cell pair. When a PE writes to the B half of its H cell (H_B), that value is transmitted to the PE's four nearest neighbors. It is then matchable in their corresponding NEWS cells. A PE can examine its neighbors' H cells by performing match operations on its own NEWS cells. Only one output driver is required per PE; the NEWS cells function only as receivers.

Move-and-add procedure. Bit-serial arithmetic algorithms can incorporate network operations. Table 5 presents the move-and-add procedure $A + B_N \rightarrow A$, in which the north neighbor's B field is added to the local field A. The first three instructions copy bit B to the special network cell H_B. If the first match is successful, the next write will set H_B to 1. Otherwise, the write in step 3 will set H_B to 0.

The remainder of the algorithm duplicates the destructive add discussed earlier, except that step 5 matches the net cell N_A instead of the local B cell. In this way, the PE obtains the value of its north neighbor's H_B cell, which will contain the copy of B_N written in the first three steps.

Combining arithmetic and communications operations in this way is faster and more memory efficient than the naive approach of copying the entire B field and performing a local addition.

Asynchronous-mode communication. The example in Table 5 used the network's synchronous mode of operation, which requires several instructions to move information from one PE to its neighbor. This works well for local communication, but results in unacceptable delays over longer distances. We can get better performance by providing a separate asynchronous communication path where gate delays–not clock cycles–limit propagation time. Illiac III[6] was an early machine to use this technique. Its network circuitry provided a flash-through mode.

Another desirable feature is connection autonomy, the ability of individual PEs to configure their net connections independently. The polymorphic torus[7] and gated-connection network[8] are representative. The fully parallel associative processor uses an asynchronous reconfigurable mesh (ARM). The ARM network provides functionality similar to the gated-connection network but is implemented quite differently.

Figure 6 shows that each of the four NEWS inputs is Anded with the B half of its corresponding network cell pair. If the

Table 5. Move-and-add procedure.

Step	Instruction	Function	Pattern A	B	C	H_B	N_A	SA contents	AR contents
1	Match		X	1	X	X	X	B	
2	Write	(SA)	–	–	–	1	–	B	
3	Write	(SA′)	–	–	–	0	–	B	
4	Match		X	X	1	X	X	C	
5	Match	(SA)	X	X	X	X	1	B_N	C
6	Match	(SA ⊕ AR)	1	X	X	X	X	A	$B_N \oplus C$
7	Write	(SA′ ∧ AR)	1	–	0	–	–	A	$B_N \oplus C$
8	Write	(SA ∧ AR)	0	–	1	–	–	A	$B_N \oplus C$

B cell contains a 1, asynchronous signals from the neighbor are enabled, and a unidirectional communication link is established. When a 1 is received from a neighbor, the H_B cell is set, and the signal propagates through the PE to all four neighbors.

Each PE can independently reconfigure its ARM links by writing to its B cells. Figure 7 shows some possible connections: a unidirectional wire (with a branch), and two bidirectionally connected regions. Once connected, the ARM network provides for simultaneous broadcasting within connected regions, using the following procedure.

- PEs configure their NEWS cells to define connectivity.
- All PEs write 0 to their H_B cells.
- Asynchronous mode is enabled.
- Senders write 1 to their H_B cells, and the network is allowed to settle.
- Each PE examines its H_B cell. If it finds a 1, it knows it is connected to a sender.

The ARM network logically Ors the outputs of multiple senders in the same region. The ability to broadcast over connected regions of arbitrary shape is particularly useful for labeling connected components and finding their corners.[3]

Note that the And and Or gates of Figure 6 are drawn only to show logical function. The actual implementation makes extensive use of precharged logic to reduce circuit complexity. The NEWS cells use only N-channel transistors and are only slightly larger than the CAPP memory cells.

Response resolution

When a content-addressable memory performs a match operation, we call the memory words that match the presented pattern *responders*. In an associative processor, the responder set may be identified by the logical combination of several match results. In either case, if more than one responder can occur, the system should have a multiple response resolution circuit. The response resolver produces summary information about the state of the array and feeds it back to the host computer and/or the individual PEs.

The ARM network performs simple Some/None response resolution. In the previous example, the H_B cell will match 1 if some senders are in the region and will match 0 if there are none. Responder prioritization and counting are examples of more sophisticated resolution tasks. The first task selects a single responder from a set of many. This procedure could be repeated to count the number of responders, or additional hardware may be provided to perform the count in time independent of the responder set's size.

Once again, the need to fit circuits on the memory pitch bounds the space of available solutions. The fastest response resolvers use tree and shower topologies[9,10] that do not lay out well in memory arrays. Linear chains are easier to lay out, but their delays increase with length. A good compromise solution is to use linear chains within each subarray, combin-

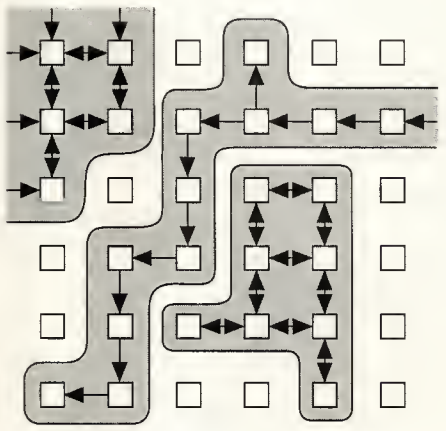


Figure 7. Reconfigurable connections of ARM network.

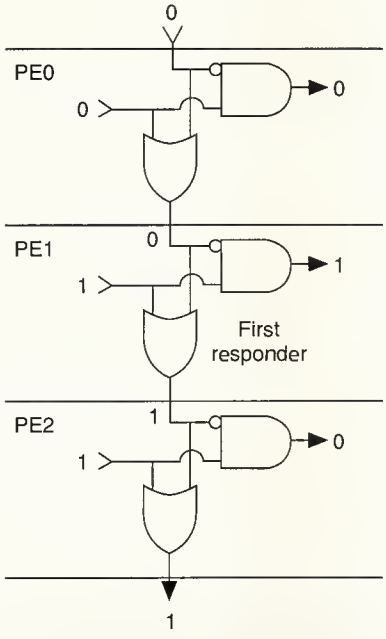


Figure 8. Prioritizing chain with Or gates.

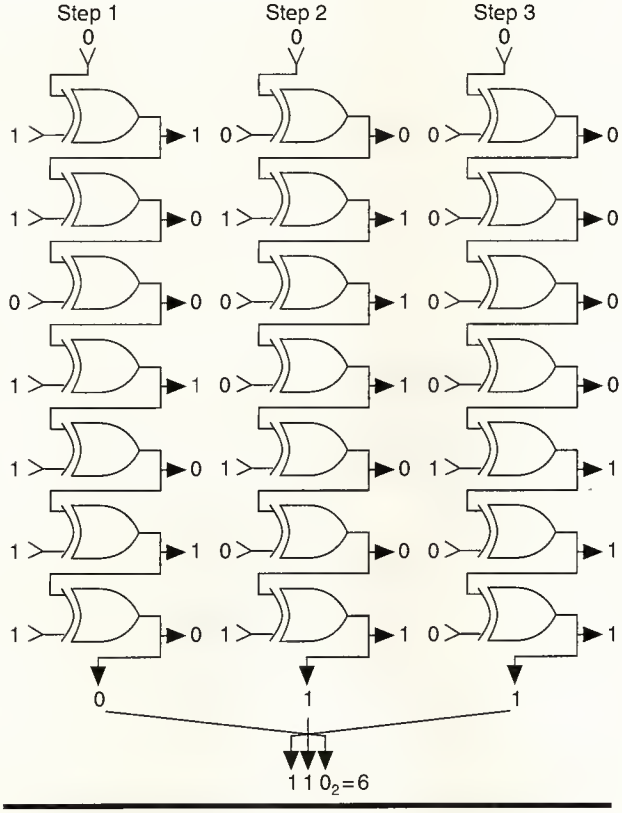


Figure 9. Counting responders with XOR chain.

ing the chain results in a tree structure.

Figure 8 shows a logic diagram for an Or gate prioritizing chain. Each input on the left side of the chain corresponds to one PE and is asserted if that PE is a responder. The output on the right side is asserted only if the PE is a responder and there are no higher priority responders. In the figure, PE1 is the first responder; it passes a 1 down the chain to inhibit PE2 and lower priority responders. The bottom of the Or chain produces a Some/None result for all PEs in the chain.

A chain of exclusive-Or gates can count responders in $\log_2 N$ steps, with N equal to the length of the chain. Figure 9 shows the three steps of the procedure for a chain of seven PEs.

The six 1 inputs on the left indicate six initial responders. The exclusive-Or chain computes the parity, and the last gate outputs a 0. This is the least significant bit of the count. Then every other responder is disabled, so three remain. The bottom of the chain outputs a 1. Finally, the even responders are again disabled, and the most significant bit 1 is obtained. Taking the bits in reverse order gives $110_2 = 6$, the number of initial responders.

Both the Or and XOR chains can be implemented with a single pass-transistor network, as shown in Figure 10. The lines $T_{1,0}$ pass 2-bit tokens from one PE to the next. The match-only cell H_A reflects the state of the chain, with each trit mapped to a 2-bit token,

$$00 \leftrightarrow X$$
$$10 \leftrightarrow 0$$
$$01 \leftrightarrow 1.$$

The 11 token is not used. The B half of the H cell serves as a responder flag. If the PE is not a responder, then $H_B = 0$. Transistors M_1 and M_3 will be on; they pass the token unchanged. However, if $H_B = 1$, then M_2 will pass $T_{1,\text{in}}$ to $T_{0,\text{out}}$ while M_4 passes $T_{1,\text{in}}'$ to $T_{1,\text{out}}$. In this way, responding PEs

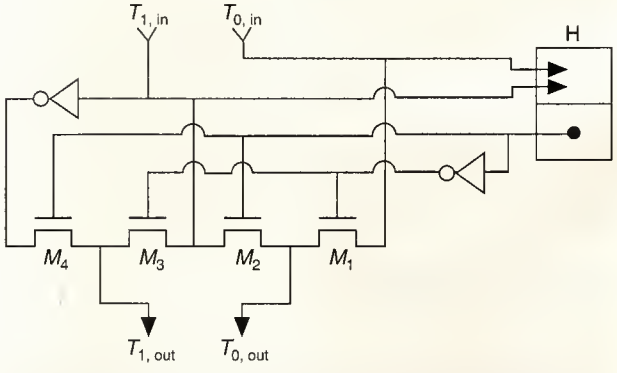


Figure 10. Pass transistor implementation of responder chain.

reverse 01 and 10 tokens, performing the Xor function. If the top of the chain is loaded with a 10 token, the bottom of the chain will produce the exclusive-Or of all the responder flags.

In priority mode, the top of the chain is loaded with two 0 bits. The 00 token propagates down to the first responder, which converts it to a 10. Lower priority responders will exchange the 10 and 01 tokens but cannot restore the 00. The chain is equivalent to a cascade of Or gates under the identification

$$00 \leftrightarrow 0$$
$$01,\ 10 \leftrightarrow 1.$$

Any PE finding a don't care (00 token) in its H_A cell recognizes there are no higher priority responders.

Note that no additional circuitry is necessary to implement the And function of Figure 8. After a PE examines its H_A cell with a match operation, it can compute the And in its function generator.

Applications

The active research in machine vision algorithms exceeds the scope of this article. Here we briefly discuss some simplified applications intended to demonstrate the utility of the associative processor in basic vision tasks, not to represent exemplary solutions to particular vision problems.

Smoothing and segmentation. Figure 11a shows an unprocessed image of a toy block. Higher level vision algorithms might require a preprocessing step to remove noise and smooth the texture. We could use a simple low-pass filter, but this would destroy useful edge information. Ideally, we want to filter only the connected regions and preserve segment boundaries.

We implemented a smooth-and-segment algorithm on a software simulator of the associative processor. The two-step procedure is a discrete-time analog of the fused resistor approach.[11]

In the smoothing step, the image is convolved with a 2D kernel such as the approximate Gaussian in Figure 12a on the next page. In the segmentation step, each pixel compares its value with its neighbors', and a segmentation flag is set if the difference is greater than a given threshold. The next smooth step will not cross a boundary if this flag is set. For example, if a pixel's east segment flag is set, it will use the modified kernel in Figure 12b in the next smooth step.

In a simulation with 8-bit pixels, the associative processor requires up to 330 match and write operations to execute both the smooth and segment steps, depending on the threshold value and the sharpness of the kernel used. We used 1,000 iterations with varying thresholds and kernels to produce the image in Figure 11b. With a 10-MHz instruction clock, the associative processor could achieve real-time performance, processing more than 30 frames per second.

Binocular stereo. Stereo matching gauges depth information by comparing two images displaced in space. It is similar to optical flow, which estimates motion by comparing

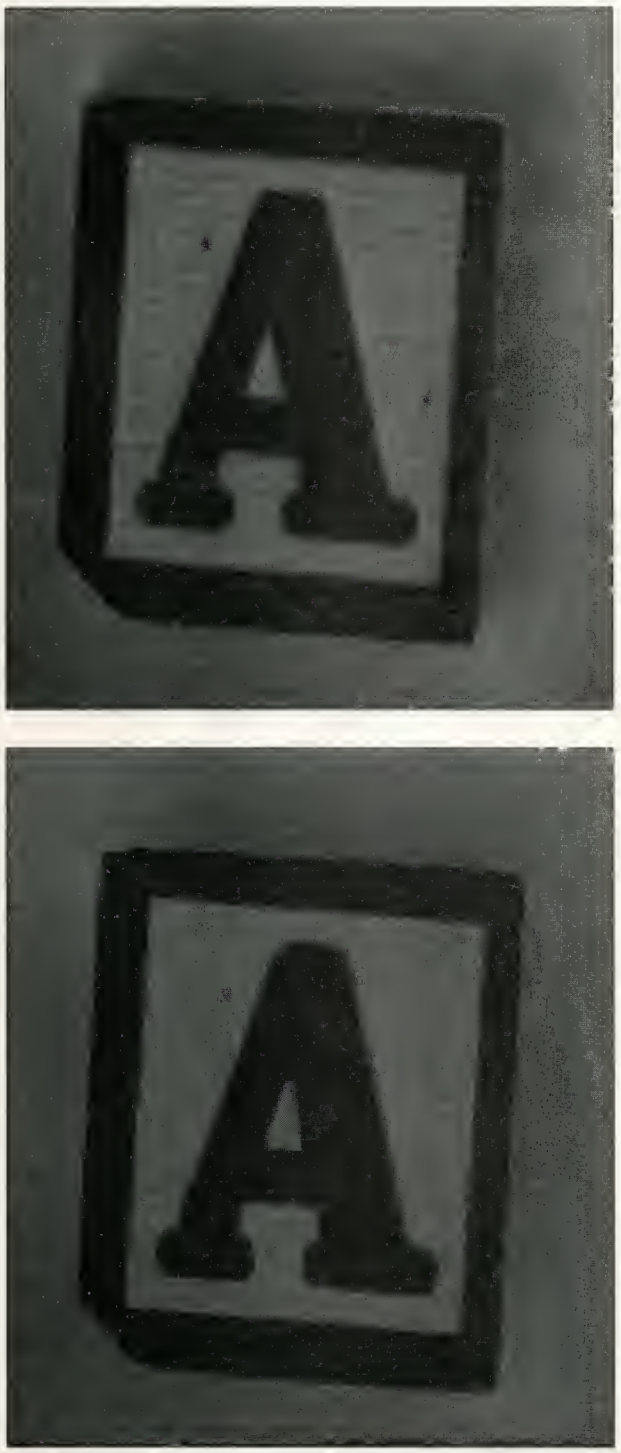
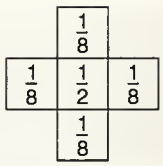
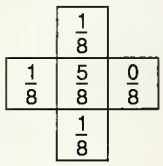


Figure 11. Original image (a) and smoothed and segmented image (b).



(a)



(b)

Figure 12. A 2D kernel (a) and a modified kernel (b).

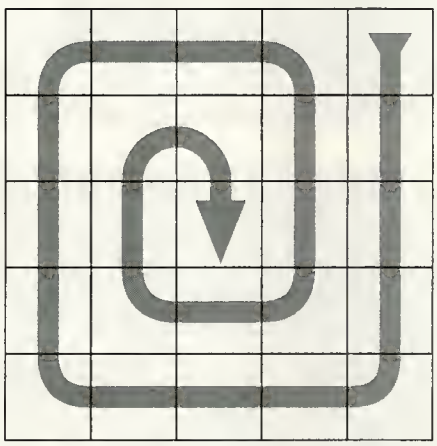


Figure 13. Accumulation path for 5 × 5 support region.

two images displaced in time.

Although they differ in their refinements, most stereo matching algorithms perform the essential task of repeatedly comparing and shifting the two images. In a typical massively parallel implementation, each PE is assigned a corresponding pair of pixels from the left and right images. The PE compares its two pixels, and examines the results of all comparisons in a support region of neighboring PEs. Then the right image is shifted relative to the left, and the process repeats with a new disparity. When the PE has tested all allowed disparities, a decision procedure determines the actual disparity at each pixel, from which the depth can be computed.

If we use a simple decision procedure (such as winner take all), then the most computationally expensive part of the procedure will be the summation of comparison results in the support region. The associative processor computes this sum using the move-and-accumulate procedure,

$$A_N + B \rightarrow A,$$

which is similar to the move-and-add described earlier. The contents of the A field are replaced by the sum of the local B field and the north neighbor's A field. The procedure requires nine match and write operations per bit.

Figure 13 shows an accumulation path for a 5 × 5 support region. Suppose the PE in the northeast corner is initialized with $A = B$. Four northerly move-and-accumulate iterations will collect all the B fields on the east edge into the southeast PE's A field. Four easterly iterations are then performed; the southwest corner will accumulate all the Bs from the east and south edges. After 24 iterations, the center PE will obtain the sum over the entire region. And because all PEs are working in parallel, every PE in the array will obtain the sum for its neighbors in the same period.

Simulation results indicate that associative processors can perform stereo algorithms similar to those appropriate for analog implementation[12] at about the standard video frame rate. This result depends on the support region area and the maximum allowed disparity, which must be chosen for each particular application. However, as in the smooth-and-segment case, image size does not impact performance.

THIS ASSOCIATIVE PARALLEL PROCESSOR architecture emphasizes high density, fine granularity, and massive parallelism. The design style is device intensive, similar to memory design rather than microprocessor design. The memory, word logic, network, and response resolution circuits all fit on pitch, and a minimum of random glue logic is required. In short, every transistor counts.

Fine granularity is achieved as each 64-trit memory word becomes its own PE. A dynamic content addressable memory cell supports fully parallel operation and allows the use of simpler word logic than is practical with a bit-serial approach. In fact, it is possible to perform useful work without any word logic at all. However, the addition of an activity register and two-input function generator significantly improves arithmetic performance with only a modest area penalty. The PEs communicate over an asynchronous reconfigurable mesh that provides a mechanism for simultaneous broadcasting over multiple connected regions. The network and response resolver use special-purpose memory cells to save area and keep the word logic simple. The target applications for the system are low- to intermediate-level machine vision and image processing tasks. Configured as a coprocessor with a desktop workstation host, the associative parallel processor may provide a low-cost, flexible alternative to the massively parallel supercomputer.

Acknowledgments

We gratefully acknowledge the contributions of Craig Keast, Phil Chu, and Shih-Jih Yao in device fabrication and application simulation. We thank Andrew Ellenberger of IBM and Terry Potter of Digital Equipment Corporation for the insights they have shared with us in many fruitful conversations. We also gratefully acknowledge Digital Equipment Corporation, IBM, and the National Science Foundation (Contract MIP-9117724) for supporting this research.

References

1. R.H. Fuller and R.M. Bird, "An Associative Parallel Processor with Applications to Picture Processing," *Proc. of AFIPS Fall Joint Computer Conf.*, Spartan Books, Washington, D.C., Vol. 27, 1965, pp. 105-116.
2. M.A. Wesley, S.K. Chang, and J. H. Mommens, "A Design for an Auxiliary Associative Parallel Processor," *Proc. of AFIPS Fall Joint Computer Conf.*, AFIPS Press, Montvale, N.J., Vol. 41, 1972, pp. 461-472.
3. C.C. Weems, "Architectural Requirements of Image Understanding with Respect to Parallel Processing," *Proc. IEEE*, Institute of Electrical and Electronics Engineers, Piscataway, N.J., Vol. 79, No. 4, Apr. 1991, pp. 537-547.
4. F.P. Herrmann et al., "A Dynamic Three-State Memory Cell for High-Density Associative Processors," *IEEE J. Solid-State Circuits*, Vol. 26, No. 4, Apr. 1991, pp. 537-541.
5. C.L. Keast and C.G. Sodini, "A CCD/CMOS Process for Integrated Image Acquisition and Early Vision Signal Processing," *Proc. SPIE Vol. 1242: Charge-Coupled Devices and Solid State Optical Sensors*, Society of Photo-Optical Instrumentation Engineers, Bellingham, Wash., 1990, pp. 152-161.
6. B.H. McCormick, "The Illinois Pattern Recognition Computer–ILLIAC III," *IEEE Trans. Electronic Computers*, Vol. EC-12, No. 6, Dec. 1963, pp. 791-813.
7. H. Li and M. Maresca, "Polymorphic-Torus Network," *IEEE Trans. Computers*, Vol. 38, No. 9, Sept. 1989, pp. 1345-1351.
8. D.B. Shu et al., "Implementation and Application of a Gated-Connection Network in Image Understanding," *Reconfigurable Massively Parallel Computers*, H. Li and Q.F. Stout, eds., Prentice Hall, Englewood Cliffs, N.J., 1991, pp. 64-87.
9. C.C. Foster and F.D. Stockton, "Counting Responders in an Associative Memory," *IEEE Trans. Computers*, Vol. C-20, No. 12, Dec. 1971, pp. 1580-1583.
10. G.A. Anderson, "Multiple Match Resolvers: A New Design Method," *IEEE Trans. Computers*, Vol. C-23, Dec. 1974, pp. 1317-1320.
11. P.C. Yu et al., "CMOS Resistive Fuses for Image Smoothing and Segmentation," *IEEE J. Solid State Circuits*, Vol. 27, No. 4, Apr. 1992, pp. 545-553.
12. M. Hakkarainen et al., "Interaction of Algorithm and Implementation for Analog VLSI Stereo Vision," *Proc. SPIE Vol. 1473: Visual Information Processing: From Neurons to Chips*, SPIE, Apr. 1991, pp. 173-184.

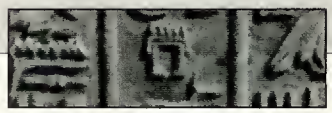
Frederick P. Herrmann is pursuing a doctoral degree in electrical engineering and computer science at the Massachusetts Institute of Technology. In his doctoral research, he is examining the circuit and architectural design of associative parallel processor systems for machine vision and image-processing applications.

Herrmann received SB degrees in electrical engineering and mathematics and the SM in electrical engineering, all from MIT.

Charles G. Sodini is an associate professor of electrical engineering at MIT. His research interests focus on IC fabrication, device modeling, and device-level circuit design, with emphasis on analog and memory circuits.

Sodini earned a BSEE from Purdue University and MSEE and PhD degrees from the University of California, Berkeley. He held the Analog Devices Career Development Professorship in MIT's Department of Electrical Engineering and Computer Science and was awarded the IBM Faculty Development Award from 1985 to 1987. He has served on a variety of IEEE conference committees and was the 1989 general chair at the International Electron Device Meeting. He currently cochairs the Technical Program for the 1992 Symposium on VLSI Circuits.

Direct questions or comments about this article to Frederick P. Herrmann, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 02139; or via e-mail at fritz@mtl.mit.edu.



An Associative Processing Module for a Heterogeneous Vision Architecture

Our heterogeneous vision architecture satisfies the computing demands of real-time computer vision by providing parallelism in three different forms. A pipeline of DSP chips initially processes signals, then our SIMD associative processor array processes images and extracts features, and a MIMD network of transputers processes extracted objects in parallel. We describe the array's VLSI implementation, the processing modes available due to the use of content-addressable memory, and the means of achieving efficient 2D interprocessor communication in the linear array. We also describe an application as a vehicle number plate recognition system.

Richard Storer

Mike R. Pout

Andrew R. Thomson

Erik L. Dagless

University of Bristol

Andrew W.G. Duller

University of Wales

A. Paul Marriott

Universidade de Averiro, Portugal

Peter J. Hicks

UMIST

Image processing and computer vision cover all aspects of extracting information from pictures. Their uses extend from producing 3D representations of a human brain to automatic, visual, industrial inspection. These applications, and the related disciplines of image generation and visualization, make strenuous demands on computers.

When an image is digitized to a resolution of 512×512 pixels, and new frames produced at 25 Hz, 6.5 million pixels must be processed every second. With this time constraint, a single processor computer, operating at a rate of 65 million instructions per second, can perform only 10 operations on each pixel. To increase the amount of processing that can be applied to all the pixels, some form of parallelism is essential.

For arithmetic operations over small neighborhoods of pixels, a pipeline or systolic array works effectively. Typically in this way, we use 2D digital signal processing chips to achieve spatial filtering by convolution. Often, we use DSP filters to reduce the amount of information in an image stream and simplify the subsequent processing. However, these chips cannot be used for nonlinear filters, such as the median filter, or operations on irregular groups of pixels, such as tracing the boundary of an object.

We can program more flexible multiprocessor or multicomputer architectures for a wider range of tasks. However, their coarse-grain, process parallelism is not ideally suited to vision tasks that require identical operations on a large number of pixels. To fill the gap between these two parallel architectures, we need a fine-grain, data parallel computer: a SIMD (single instruction, multiple data) processor array. For example, by processing 1,000 pixels at a time with a thousand 20-MHz processing elements, we gain sufficient time to perform 3,000 operations on each pixel.

All three of these computer architectures have their place in computer vision systems. A vision task is often divided into stages in which each stage produces fewer but more complex data objects for the subsequent stage. The different stages can then be handled by different types of processors.

One such approach led to the development of pyramidal designs such as the Image Understanding Architecture (IUA).[1] The processing pyramid features three layers of parallel processors corresponding to three stages in typical vision processing. At the base of the pyramid, a large array of simple processors performs pixel-oriented tasks. This layer produces regional information—for example, concerning edges or texture in the

0272-1732/92/0600-0042$03.00 © 1992 IEEE

image—which then passes to a smaller array of more powerful processors. The smaller array associates these features with the objects in the image that are to be recognized by the highest layer.

The IUA is a large machine of fixed topology intended to cope with all vision tasks effectively. An alternative approach is to develop a set of vision computing modules that can be combined in a more flexible way. Designers can assemble an optimum configuration of modules for the set of vision tasks required in a particular application. A small, heterogeneous vision architecture (HVA) implements the application.

HVA

Our heterogeneous vision architecture comprises four different types of modules:

- DSP for linear filters and other simple, pixel-oriented tasks. Currently, we use Inmos A110 2D DSP chips.
- An associative processor, SIMD array for nonlinear filters, morphological, and other region-based operations. We plan to build these modules using the GLiTCH associative processor.
- A multicomputer, MIMD (multiple instruction, multiple data) array for manipulating model databases and directing the operation of the other modules. We use Inmos transputers for this module.
- A programmable frame store for buffering image data between processing modules. These custom frame stores can deliver any rectangular image patch required by a processing module.

Figure 1 shows an example of a combination of these modules.

Each module contains at least one transputer, which directs communication with other modules using the transputer serial links. The whole machine, then, can be seen as an array of transputers, some with specialist functions. Image data passes between modules using the 8-bit, point-to-point, Maxbus standard.[2] Some modules may have more than one video input and output, allowing one-to-one, one-to-many, or many-to-one video connections.

The highest level vision process, in one of the transputers, delegates tasks to other modules according to their specialization. As processing continues, it may modify these tasks, for example, by instructing a processing module to operate on only a particular part of the image or to make adjustments to filter coefficients. This vision process also controls the path of an image through the various processes.

The associative processor module

This module processes a rectangular patch of an image held in one of the programmable frame stores. The processed patch may pass to a second frame store or return to its source. The associative processor is a 1D, SIMD array of GLiTCH processing elements (PEs) that are usually assigned one image pixel each. We are building GLiTCH chips (which contain 64 PEs each) plus a microcode sequencer, pattern store, data-routing network, and controlling transputer to form the array. (See Figure 2 on the next page.)

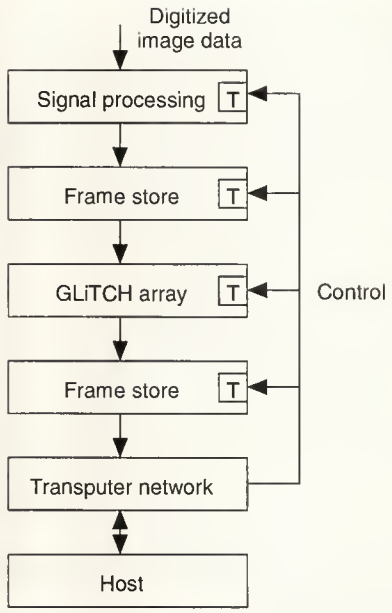


Figure 1. A heterogeneous vision system. T indicates transputer.

The controlling transputer decides how to process the patch, acting on information received from other modules in the HVA and on the results of previous processing. This transputer calls GLiTCH microcode subroutines and supplies parameters via the pattern store, which is mapped into its memory space. The transputer also decides the size, shape, and position of the image patch and sends the appropriate request to the frame stores. The shape of patches may be chosen according to the pixel neighborhood requirements of the algorithm, or to process a particular part of the image containing items of interest.

While the GLiTCH subroutine is active, the microcode sequencer provides instructions for all parts of the module. It implements loops, subroutines, and branches on the status of internal counters and flags from the PEs, frame store interface, and scalar register. The scalar register holds and tests scalar operands for scalar-vector operations, such as multiplying all pixel values by a filter coefficient. The subroutine may call upon processes in the transputer to provide scalar processing. Operands and their results transfer between the array and the transputer via the pattern store.

An 8-bit video shift register (VSR) running through all the PEs in the array loads the image patches as they arrive from

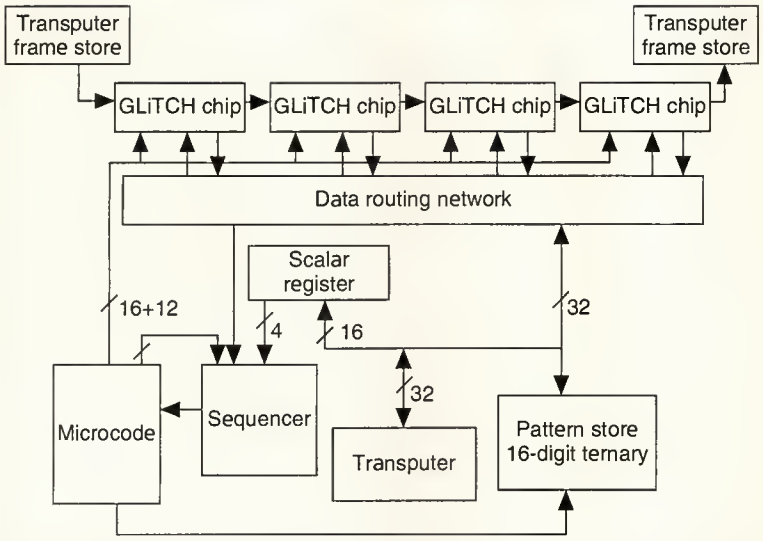


Figure 2. A GLiTCH array.

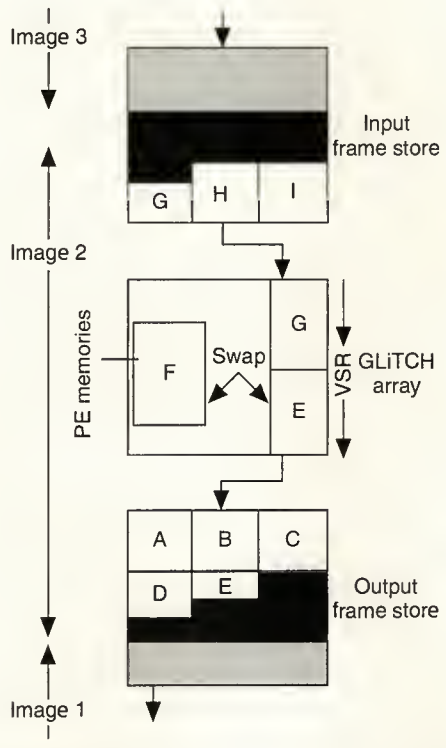


Figure 3. A patch-processing pipeline in a GLiTCH array. Letters A-I indicate consecutive patches of the image.

the frame store (see Figure 3). At any time, typically three patches are present in the system, one being loaded into the GLiTCH VSR, one being processed in the array, and one being output from the VSR. When processing and loading are both complete, the patch in the array exchanges with the patch in the VSR, and the operation repeats. The current GLiTCH design permits a maximum clock rate of 20 MHz, but the VSR operates asynchronously to the array at clock rates of up to 40 MHz.

The GLiTCH chip

Based on our experience with the simulator, we made the following design choices. Each GLiTCH chip contains 64 one-bit PEs, each with 64 bits of content-addressable data memory (CAM) and 4 bits of subset memory. In addition, GLiTCH contains an 8-bit-wide VSR, an instruction decode ROM, and some pattern broadcast logic (PBL). Figure 4 shows the chip floor plan.

Each PE comprises a full 1-bit arithmetic and logic unit, three registers (tag, carry, and subset or activity), and some read/write control logic for its CAM (see Figure 5). All PEs operate together in lockstep, under the SIMD paradigm, but the subset register provides local control. The value in this register can selectively disable the PE from writing data to memory, thus allowing different PEs to execute effectively different operations over time. This option implements conditional execution.

Ternary patterns in GLiTCH support both pattern matching and fetching arguments for bit-serial operations. The pattern store supplies these patterns, in conjunction with an instruction from the microcode memory. The pattern store also supplies two separate 8-bit patterns that the PBL independently routes to match the appropriate columns of the CAM. The results from the matching pass to the PE. Say the two patterns are each one significant bit long, and the other bits are don't cares (match with either a 1 or 0 stored). Then, the PE can determine the contents of the two specified CAM cells and use these values for bit-serial arithmetic and logic operations. It can also access the tag register contents of its immediate neighbors.

The full pattern-matching capability of the system could support graph traversal, set manipulation, and database operations among others.

Every time a match takes place in the main CAM, a match also occurs in the subset CAM. The pattern for this comes from the microcode store with the instruction. The result of this match feeds directly into the subset register in the PE. The four subset bits can partition the array into different groups of PEs according to the programmer's own criteria.

Each processor's data CAM can also be read and written via the data bus; the subset bus allows only writing and matching. Though any number of PEs can be written simultaneously, they may only be read one at a time. The programmer specifies which PEs are to be written or read by a logical combination of tag, carry, and subset register bits or by selecting the first PE with a set tag bit.

Both the main CAM and the subset CAM use a dynamic memory cell to keep area low. We determined the CAM sizes (64 bits and 4 bits) by simulation, which showed that 64 bits per PE sufficiently supported integer operations on 8-bit pixels.

After each instruction executes, the PEs generate two signals, carry reply and tag reply. These wired-Or signals reflect all the tag and carry register contents in the array and feed to the microsequencer. They can be used to skip sections of microcode when no PEs contain relevant data, or to terminate loops when all PEs have reached some new state.

When necessary, data passes to distant PEs bit serially from tag register to tag register. In each machine cycle, each PE passes a tag bit from one neighbor and receives a tag bit from the other, until the bits reach their destination. For long-distance communication, groups of 32 tag bits pass as 32-bit words from each chip, through the data-routing network, to their destination chips.

All PEs connect to an address/pattern-generating network that runs through the whole array, connecting chip to chip. This network can number individual PEs (useful when processing image data to determine where in the image a pixel lies), for counting the number of PEs with tag registers set, for identifying individual PEs for reading data to the pattern store, for divide-and-conquer algorithms, and for certain types of set operation.

A team at UMIST (the University of Manchester Institute of Science and Technology) designed the GLiTCH processor array chip using 2-micron rules in full-custom CMOS technology. It is approximately 9 mm×10 mm in size, contains 90,000 transistors, and dissipates 1W of

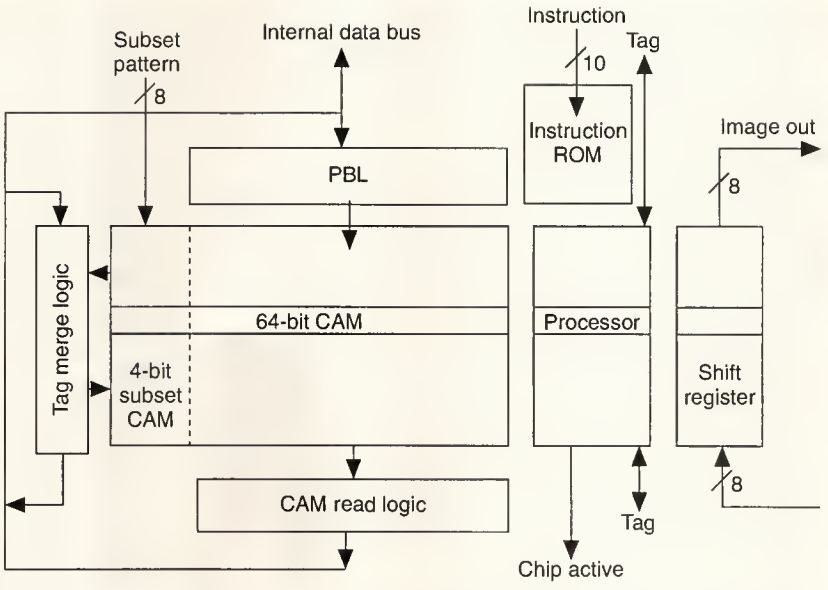


Figure 4. Floor plan of the GLiTCH processor array chip. PBL indicates pattern broadcast logic.

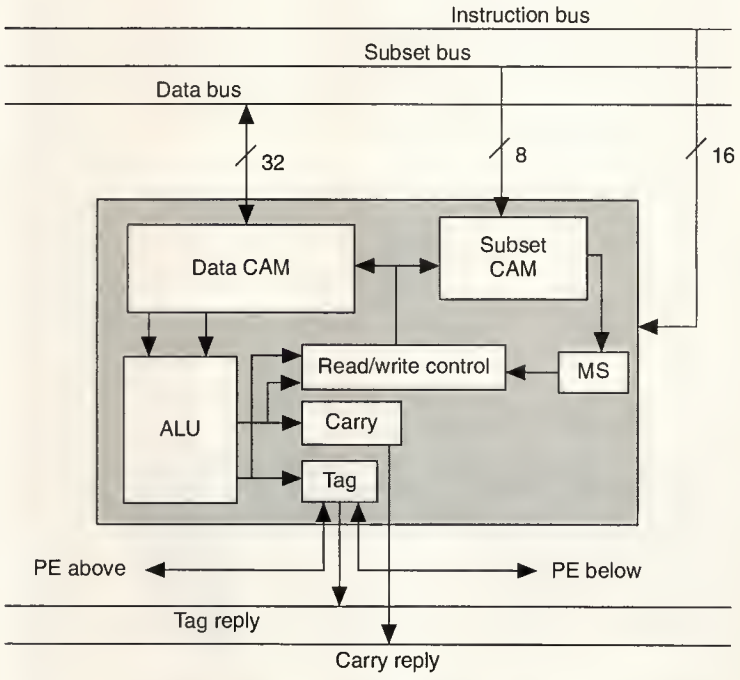


Figure 5. A GLiTCH processing element.

power. Its maximum clock speed is 20 MHz. For further information see Duller et al., Thomson, and Storer.[3-5] We are currently redesigning the chip to allow rapid synthesis of different GLiTCH variants, as described in Marriott et al.[6]

Programming GLiTCH

We write GLiTCH programs in an assembly language specially developed for the machine. This language has a format similar to C and provides a simplified interface to the system. It is implemented as a collection of C function calls, one for each assembler statement.

CAM matching and writing. The match and write operations are fundamental to an associative processor. GLiTCH allows the programmer to match or write two 8-bit fields to the contents of the CAM of each PE in one instruction cycle. The programmer can also match a subset pattern to specify groups of PEs to take part in or ignore subsequent commands. The match patterns appear as ternary strings, for example,

```
match (0, "110x01x0", 8, "11xxxx01", "1xxx");
```

The rightmost digit of the first pattern, 0, is matched against column 0 of the CAM. The digits to its left are matched against columns 1 to 7. Similarly, the rightmost character of the second pattern, 1, is aligned to column 8. The third pattern is the 4-digit subset pattern—in this example, only one of the digits in the pattern is significant, the rest are all don't cares. This example is unusual as the two patterns are adjacent; they could be positioned at any column over the CAM, and don't cares in one pattern can overlap with data in the other.

Similar to the match command is

```
write_all (tag, 24, "11x10", 60, "10110xx1", "1x0x");
```

This instruction writes the patterns specified in all PEs that have their tag registers set. The second pattern wraps around, to overwrite columns 0 and 3.

1xxx	...xxx1xxx0	Broadcast pattern			
(a)					
subset bits	data bits	tag	carry	subset	
1101	...11100110	0	0	1	
0000	...00110011	0	1	0	Array of
1000	...10100110	0	0	1	processing
0111	...10100110	0	0	0	elements
1010	...10110110	1	0	1	
1100	...10010100	1	0	1	
(b)					

Figure 6. An Add operation (a) produces a result in all the tag and carry registers (b).

Another example of the write instruction is

```
write_local (subset, tag_above, 16, "1", "");
```

We named this instruction write_local because it writes data held in the PE, rather than globally broadcast data. A variant of the write command exists for this latter case.

The first parameter, subset in this case, determines which PEs will execute the write command. Here all PEs with a 1 in their subset match register will write data to the CAM; the others will leave their CAM unchanged. Other possibilities for this parameter are all, tag, carry, or the negation of one of these.

The second parameter specifies what data will be written to the CAM. In this case the choices include tag, carry, image_in, tag_above, and tag_below. Image_in selects data from the VSR; the latter two cases select tag values from the neighboring PEs and enable PEs to work together. Writing a 0 instead of a 1 inverts the data.

The third and fourth parameters contain the position and data to be written back; the fifth pattern contains the subset pattern to be written, which is empty in this case.

Modes of processing. The combination of content-addressable memory and a one-bit processor allows the programmer to choose different modes of processing as appropriate to the problem.

Bit-serial arithmetic. Being 1-bit devices, the GLiTCH PEs execute multibit operations one bit at a time. The CAM cell employed in GLiTCH has two match lines coming from it, a match 1 (MD1) and a match 0 (MD0). Broadcasting two 1-bit patterns, a 1 and a 0, allows the contents of two CAM cells to be determined. With this information the PE can calculate a 1-bit arithmetic or logical operation in just one cycle; writing the result back requires a further cycle. Comparable systems using RAM rather than CAM take two cycles to fetch the operands, and may then need a further cycle to perform the operation, making them up to twice as slow. If, unlike GLiTCH, the on-chip data is not available, arithmetic operations may be slower still, depending on the access time of the memory. For GLiTCH, access takes only 10-20 ns.

The PE contains two arithmetic registers, tag and carry. In addition and subtraction operations, the tag register loads the result and the carry register loads the carry. The GLiTCH assembly language contains a large number of arithmetic and logical instructions that provide both vector-vector and scalar-vector computations. It is not usual for the programmer to write bit-serial routines; a library of commonly required functions is available. This library contains instructions to support both signed and unsigned fixed-point and floating-point calculations and comparisons. Figure 6 shows the effect of the assembly language add instruction

```
add (0, 4, "1xxx");
```

Lookup-table processing. An alternative to bit-serial computation on bit fields is to use the data CAM as a lookup table (LUT).

The array controller broadcasts all possible values of the operands sequentially, each time writing the corresponding value of the result into the PEs that record a match. With GLiTCH, this operation requires two machine cycles per operand value, for operands and results of up to 16 bits.

In general, a LUT operation on an n-bit operand (or n-bit word that combines a number of smaller operands) requires 2^{n+1} cycles, the equivalent of $2^{n+1}/2r^2$ r-bit, bit-serial additions per bit of an r-bit result field. When n equals 8, a LUT operation is equivalent to four 8-bit additions for each bit of an 8-bit result or one 16-bit addition for each bit of a 16-bit result. Thus LUT is a useful technique when several bit-serial additions, subtractions, or comparisons would be required to calculate each bit of the result, as in successive approximation or summation of a series.

For example, to calculate the 16-bit square root of an 8-bit field by successive approximation requires 760 machine cycles, whereas to broadcast all 256 possible 8-bit values and their corresponding 16-bit results requires only 512 machine cycles. In addition, the bit-serial algorithm requires working space in the CAM, but the LUT version does not.

Bit-parallel, word-serial arithmetic. This type of arithmetic uses a number of PEs to perform one arithmetic operation. Each PE holds a single bit from each operand. This type of mode helps in the massive parallelism of processor arrays that cannot be used because of the lack of operands. In reduction functions—summing an array of values, for example—cascaded addition leads to reduction in the available parallelism. Using bit-parallel arithmetic increases the number of PEs that can be used, and consequently the operation can be performed more quickly.

For example, to add two sparse vectors of 32-bit numbers, groups of 32 PEs hold one number from each vector and one bit from each operand in each PE. Each PE adds its two bits and the carry generated by its neighbor in the group. In effect, the carry bits produced in the PEs holding the least significant bit in each group must then be "rippled" through the rest of the group.

Reading and multiple-match resolution. The processor can read up to 16 bits of data from one selected PE in the system, saving the data in the pattern store, the system's global memory. From here, data can either be broadcast to the whole array, passed to the scalar register for a scalar-vector computation, or read by the host transputer for a purely scalar operation.

The format for the read instruction is

```
read_cam (carry, 48, 16);
```

which reads 16 bits of data, starting at column 48, from the PE with its carry register set. Only one PE must be selected; the programmer must ensure that a reliable method is used, otherwise contention can occur. This example assumes that only one PE in the array has its carry register set; this assumption permits determination of which PE executes the instruction. Other options for the PE selection are tag, tag_above, and tag_below.

Two MMR (multiple match resolution) functions help in identifying a single PE. One function trickles through the array, starting at the top, until it finds a PE with its tag register set. It sets the carry register in this PE *only*. This function can read out a series of data from the array, one at a time.

The second function generates repeating sequences of 1s and 0s in the tag registers. It too trickles through the array. This function can generate a unique address for each PE, which is useful if it is known which PE will contain the result; then its address can simply be matched.

The hardware also supports a number of program control constructs. These implement conditional and unconditional branches; for, do, and while loops; and subroutines. The assembler also implements Case statements. Further program constructs support interprocessor communication, variable declaration for use in macros, setting of breakpoints, and production of trace files that can be used to analyze program efficiency.

Data-routing network design

The way PEs communicate vitally affects the efficient performance of any processor array. The organizational time (time required for data communications) of many algorithms will dominate if the communication highways are not of sufficient bandwidth. Experience with using a linear array for image processing led us to conclude that a 1D network was inadequate for this purpose. A 2D network, however, would require many pins per package (32 for an 8×8 array of PEs), make for complicated interboard wiring, and impose constraints on possible system sizes. Some alternative was necessary, combining the strong points of both but without the drawbacks.

After concluding that the data pins would be used for data movement as well as matching, writing, and reading, we produced a design for a data-routing chip, which is shown in Figure 7 on the next page.

The data-routing chips provide a reconfigurable data path between the GLiTCH chips and the data store but introduce a further pipeline delay into the system. Four identical data-routing chips are associated with four GLiTCH chips, each router chip handling 8 bits of the 32-bit-wide data buses between the GLiTCH chips and the pattern store. This arrangement helps to keep the number of circuit board tracks down since many connections are made internally. Even so, each circuit board with four GLiTCH chips and four data routers will be very complicated to lay out.

Chip specification. The router chips sit between the data store and the GLiTCH chips, and introduce a one-cycle de-

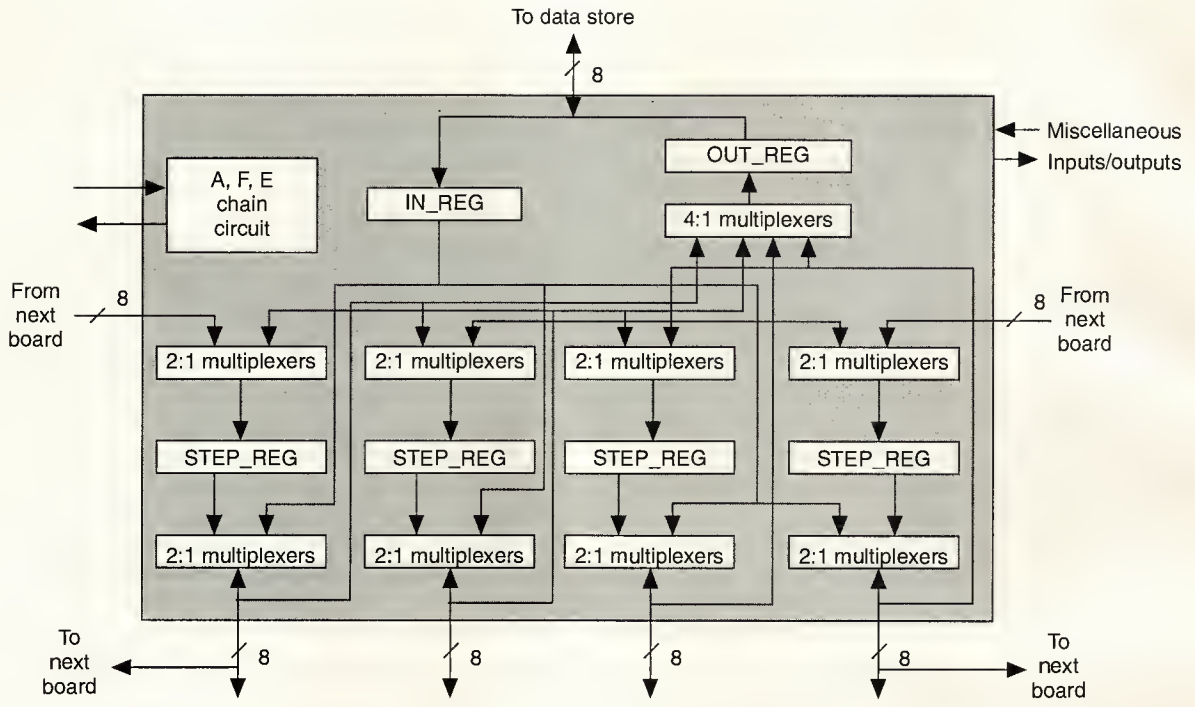


Figure 7. Architecture of the router chip.

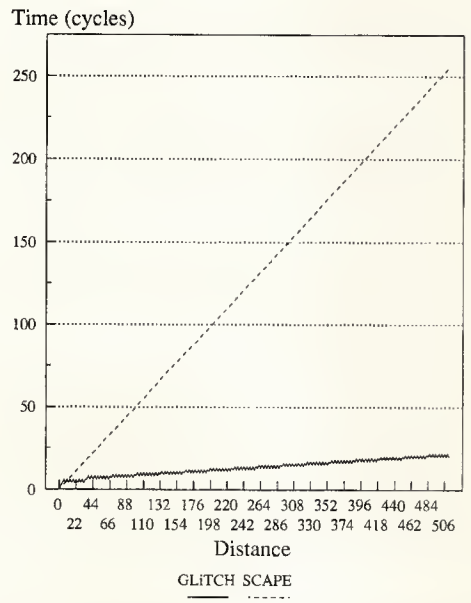


Figure 8. Comparison of data movement on GLiTCH and SCAPE.

lay. Designed to operate at up to 20 MHz, they support the following three main data transmission functions:

- broadcasting of global data from the pattern store to all GLiTCH chips,
- relaying a single datum from one GLiTCH chip back to the data store, and
- shifting data from all GLiTCH chips in parallel.

Shifting data. The first two functions are straightforward to implement, but the shifting task needs further explanation. The four data-routing chips associated with each four GLiTCH chips allow 32 bits of data per GLiTCH chip to be moved every cycle. For a 16-chip system, this approach gives a huge, 10-Gbits/s bandwidth, which scales linearly as the system size increases. The three stages in shifting data along the GLiTCH array are loading data from the GLiTCH chips, shifting it along the array, and writing it back.

Data passes from all GLiTCH chips in parallel through a multiplexer into the STEP_REG to the left or right. From here, it shifts from register to register within the router chip, each step moving it one GLiTCH chip (64 PEs) up or down the array. Data output from the last STEP_REG in a chip feeds into the corresponding chip in the next group of four. (Throughout all this data movement, the GLiTCH chips sit idle.) When the data finally arrives at its destination, it is

output through a multiplexer and the GLiTCH chips latch it. This sequence then repeats to move the second 32 tag bits.

Network performance. For GLiTCH to succeed as a real-time image processing system, high data-movement bandwidth is essential. Further, the 1D nature of the array must not compromise that high bandwidth by requiring unduly large shift distances.

Earlier we stated that a 16-chip (1,000 PEs) GLiTCH system has a data movement capacity of 10 Gbits/s. In addition, when data steps from one routing chip to the next, it is in effect moving 64 PEs along the array. If we compare a 1,000-PE GLiTCH array with a similarly sized 32×32 PE 2D array, we can see that one step along the GLiTCH array compares with two vertical steps in the 2D array. Our comparison is tempered by the fact that GLiTCH can only shift 32 bits of data for each 64-PE chip at a time. The operation must be repeated, doubling the time and bringing parity with the 2D array in the example just cited.

However, GLiTCH is not obliged to shift data in exact multiples of 64 PEs. By making use of the barrel shifter in the CAM and some special logic in the PBL, it can shift any multiple of four, taking the same amount of time as for the next higher multiple of 64. Thus, a shift of 48 PEs takes as long as a shift of 64, an important point when we compare it with a 2D array. Here, a shift of 48 maps to a vertical shift of one and a horizontal shift of 16, a total of 17 cycles; but GLiTCH still needs only to move the data one *chip*. The time GLiTCH takes is thus a fraction of the time required for the 2D network.

To illustrate these findings, we compared three versions. The first version uses SCAPE, a true 1D array. SCAPE can move data up to two PEs per cycle; we assumed both SCAPE and GLiTCH would have the same cycle time. Under these conditions, GLiTCH is an order of magnitude faster than SCAPE. A graph of the results appears in Figure 8.

The second comparison was the same as that example, but compared GLiTCH and a 32×32 2D array. On average, GLiTCH is a few cycles faster in this comparison, though on the purely vertical shifts it does not perform as fast since its setup time is significant. Figure 9 shows this result.

Finally, we compared GLiTCH with a 64×64 2D array. While GLiTCH performs faster on average for shorter length shifts, its comparative performance steadily worsens as the shifts get longer. This result appears in Figure 10.

In conclusion, the GLiTCH data movement network outperforms 2D arrays for sizes up to 32×32. It retains 1D connectivity making it easy to wire up, easy to extend, and keeping the GLiTCH package pin count down.

An application study

Bristol University designers built a machine for reading British vehicle number plates (registration or license plates) in video images using one transputer and standard image

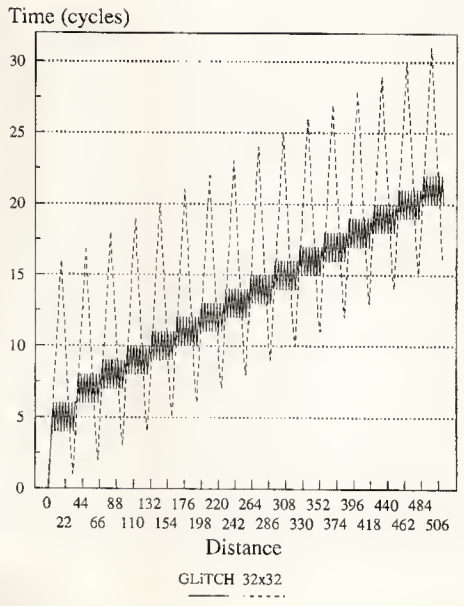


Figure 9. Comparison of data movement on GLiTCH and a 32×32 2D mesh.

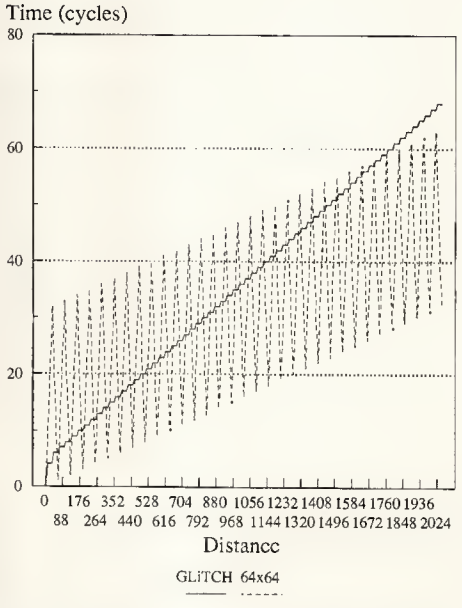


Figure 10. Comparison of data movement on GLiTCH and a 64×64 2D mesh.

processing techniques.[7] We have run this, and other applications, on a simulation of a GLiTCH array to measure the chip's image processing performance. The case of the number plates provides some illustration of the variety of techniques for which GLiTCH is suited.

The GLiTCH program follows the general solution used on the transputer system, substituting some techniques that are more efficient on SIMD machines. The transputer system reads most number plates in 1.5 to 2 seconds. Faster processing, using a GLiTCH array, would allow several attempts at reading difficult plates as the vehicle moves across the image field, even if the number plates are digitized at increased resolution.

British plates come in a standard size with up to seven, black, alphanumeric characters on a white or yellow background. Some variation in the style of the characters exists; some may have four or five variants. Typically, the number plates occupy one third of the width of a video frame digitized to 256×256 pixels in 256 gray levels. GLiTCH reads them using the following sequence of processes:

- Reduction of the 256 gray levels to two using a simple adaptive threshold that separates the black characters from their brighter background;
- Removal by a 3×3 median filter of small objects from the binary image and smoothing of the edges of the larger objects;
- Numbering by the region-growing operation of all the black objects in the image, a process that labels each pixel with the number of the object it belongs to;
- Determination of the bounding rectangle of each black object in the image;
- Acceptance of those objects that have a suitable size and aspect ratio as number plate characters; they are resized to be the same size as the set of reference characters; and
- Identification of characters using different techniques. 1) GLiTCH counts the number of holes in each candidate character using the region-growing operation on white objects within the characters. It then compares each character with reference characters having the same number of holes. 2) The GLiTCH array models a single-layer neural network trained previously on the reference character set.

Figure 11 shows an example number plate image in several stages of processing.

The GLiTCH array chosen for this application comprises 32 GLiTCH chips (2,048 PEs), a 4-Mbyte processor RAM (16 Kbits per PE), and 32 data-routing chips. This array together with the GLiTCH controller, image digitizer, and two programmable frame stores fit within a small VME card cage. This array size enables the task to be completed at video frame rates.

The adaptive threshold and median filter. A simple adaptive threshold converts the gray-level image containing the number plate into a binary image with black characters on a white background. GLiTCH then takes the threshold value for each pixel as the mean of the pixel values in the 11×11 neighborhood centered on the pixel. The process renders a pixel white if it has a brightness above this threshold and black otherwise. This calculation uses bit-serial arithmetic on image patches of 32×64 pixels and requires 1,193 GLiTCH clock cycles for each.

If the VSR loads one pixel per clock cycle, it will take 2,048 clock cycles to load the next image patch. In fact, the VSR loads asynchronously with the GLiTCH processors and may use a faster clock. In the prototype, however, both use the same clock, and the PEs will be idle for 44 percent of the time. To make better use of the time, we combined the adaptive threshold and subsequent median filter for each patch.

In a binary image, a median filter involves simply counting the number of white (or black) neighbors around each pixel. The median value will be white if more than half the neighbors are white, and black otherwise. Our 3×3 median filter takes 161 clock cycles for each patch.

The boundary pixels in each patch cannot be processed without all their neighbors, so the filter must overlap the patches to make sure that all the pixels are processed. Combining an 11×11 adaptive threshold with a 3×3 median filter requires an overlap of 12 pixels. With 2,048 PEs, the input frame store delivers 65 overlapping patches of 32×64 pixels. The output frame store discards the invalid results from the overlapping areas as it pieces together the processed patches that come out of the GLiTCH array. Figure 11 shows the binary image resulting from the combined adaptive threshold and median filter.

Even when the adaptive threshold and median filter are combined, the total processing time still takes less than the total video I/O time. This is shown in the output from the GLiTCH program profiler, which also gives the time spent on inter-PE communication:

Array busy time	83330 cycles	62% of total time
includes:		
communication time of	38935 cycles	47% of busy time
processing time of	44395 cycles	53% of busy time
Array idle time	50830 cycles	38% of total time
VSR busy time	134160 cycles	
VSR idle time	0 cycles	
Total time	134160 cycles	

To reduce the processor idle time, GLiTCH could perform the next part of the number plate recognition process on each

Figure 11. Stages in the automated reading of a vehicle number plate: the digitized video image (top left), after the adaptive threshold and median filter (top right), the selected bounding rectangles (bottom left), and the reference character set (bottom right).

patch while it is in the array. However, in this case, performing the region-labeling operation on the patches as they are filtered is not desirable, because the optimum patch shape for the filtering and region labeling are not the same. As a result, we lose 50,830 cycles of potential processing to allow the filtered image to be assembled in a frame store and reloaded in a different format. Using a faster clock for the VSR or performing this function in a second GLiTCH module with a smaller GLiTCH array would avoid this situation. In an array with 1,024 PEs the patch loading time more nearly matches the patch processing time, thus reducing the total time for the operation even though more patches are required.

Region labeling. Lotufo et al.[7] use a boundary-following algorithm to locate the likely number plate characters in the

Operations on neighbors' registers do not require explicit inter-PE communication.

binary image. This algorithm on an SIMD machine would only make use of a very few PEs at a time, those that were at the end of the boundary or boundaries being followed. To make use of the parallelism in GLiTCH, a connected component-labeling algorithm assigns unique labels to all four-connected, disjoint, black regions in the image. With this technique, the region label spreads quickly across its area, not just around the outside.

When processing the image in patches, GLiTCH must resolve labels across patch boundaries. If the patches are arranged as vertical strips, the algorithm will require checks only on the horizontal consistency between the labeled regions. With a system of 2,048 PEs, we use 32 non-overlapping patches, 8-pixels wide×256-pixels high. Reorganizing the image requires only an instruction to the programmable frame store holding the output from the combined adaptive threshold and median filter.

The region-labeling operation in each patch uses the bit-parallel matching and multiple match resolution facilities of the CAM. The time it takes depends on the number and size of the regions to be labeled. In the example in Figure 11, on average, about 20 regions in each patch take about 360 cycles each to label. The time to label each patch is generally longer than the time to load the next one, so the PEs are only idle while the first patch is loaded into the VSR:

Array busy time	231534 cycles	99% of total time
includes:		
communication time of	36353 cycles	16% of busy time
processing time of	195181 cycles	84% of busy time
Array idle time	2048 cycles	1% of total time
VSR busy time	68112 cycles	
VSR idle time	165470 cycles	
Total time	233582 cycles	

As PEs have access to the tag registers in their immediate neighbors, operations on neighbors' registers do not require explicit inter-PE communication. This fact accounts for the apparently low proportion of inter-PE communication time logged during the labeling operation.

The 11-bit labels for the pixels in each vertical patch are stored in the processor RAM as they are generated. To resolve the region labels, GLiTCH copies pairs of patches to CAM and compares the labels along their common boundary. The algorithm passes alternately leftward and rightward across the image until all the labels in adjoining patches match.

Typically, the example number plate requires two passes through the label image, using a total of 130,000 cycles. The frame store requires no new data, so all this time is useful processing:

Array busy time	129576 cycles	100% of total time
includes:		
communication time of	38160 cycles	29% of busy time
processing time of	91416 cycles	71% of busy time

Finding the bounding rectangles. The algorithm uses the size and aspect ratio of the bounding rectangle of each region to judge whether the region is likely to be a number plate character.

GLiTCH copies each patch, which now consists of an 11-bit label in each pixel position, from RAM to CAM in turn. For each patch, the GLiTCH array calculates the extreme x and y addresses for the parts of each region in that patch. The controlling transputer, which compiles a table of acceptable regions and their bounding rectangles, reads these addresses from the array. The transputer updates this table in parallel with the GLiTCH operations on the next patch.

In the example image, the transputer selects seven candidate characters from the bounding rectangles of the 157 objects detected by the array. Figure 11 shows the selected bounding rectangles superimposed on the binary image.

In each patch, the algorithm identifies the members of a region with a bit-parallel search and uses a bit-serial minimum and maximum algorithm to find the extreme x and y addresses. The example image used a total of 93,219 cycles; no inter-PE communication or data was loaded via the VSR.

From this point on, we no longer need to process the whole image. The frame store containing the binary image is programmed to deliver only the portions of the image containing the candidate characters. As the characters are expected to be smaller than 32×64 pixels, a single patch for each of the candidate characters suffices.

Scaling the characters. To resize the unknown characters so that they are the same size as the reference set, the controlling transputer calculates the scaling factor needed to bring each character's bounding rectangle to the standard size. For each character in turn, the transputer loads into the array the 32×64 image patch, which has the bounding rectangle in the top left corner. The patch is resampled to the required scale and then classified by template matching.

The average resampling time for each patch in the example image is 729 cycles, of which 23 percent involves inter-PE communication. We included this time in the analysis of

the character classification method we describe later. The resampling algorithm makes much use of the bit-parallel matching and writing capabilities of the CAM. If bit-serial operations were the only ones available, the execution time of this part would increase by about 70 percent.

Character classification by template matching. The reference templates are 16×32-pixel, binary characters, as shown in Figure 11. After resampling to this size, the algorithm compares four copies of each unknown character with four templates at the same time, using 512 PEs for each comparison. Each unknown character is compared with each reference character that has the same number of holes. The reference character with the fewest different pixels produces the best match.

The region-growing technique described earlier counts the holes in the characters. Holes are white regions in the character that do not touch the bounding rectangle. As the character fills the array exactly, it is processed in one patch with no need to resolve the region labels. Counting holes in the example image requires an average of 7,000 cycles for each character.

The technique for counting the number of different pixels uses bit-parallel arithmetic. A set tag register in the corresponding PE represents each differing pixel; that is, 2,048 one-bit values must be summed. Pairs of adjacent PEs combine their values so that each pair now holds a 2-bit number, one bit in each PE. The 2-bit values from adjacent pairs are overlapped and added, in bit-parallel arithmetic, to form a 3-bit value (0 to 4) across four PEs. This process repeats until just one value, distributed across the topmost PEs, remains in the array. The controlling transputer reads this total via the CAM barrel shifter and PBL. The example image requires about 4,000 cycles for each character, of which about 50 percent involves inter-PE communication.

The timing analysis of the complete number plate reading program is as follows:

Array busy time	619310 cycles	92% of total time
includes:		
communication time of	133165 cycles	22% of busy time
processing time of	486145 cycles	78% of busy time
Array idle time	54926 cycles	8% of total time
VSR busy time	216720 cycles	
VSR idle time	457516 cycles	
Total time	674236 cycles	

For a prototype GLiTCH system with a 20-MHz clock, this represents 33.7-ms total time for each image frame.

Character classification using a neural network. Recently, we investigated a neural network as an alternative method of character recognition. For the number plate characters we used a one-layer Perceptron that was trained using the Delta rule.[8] The training, on a software simulation of the network, gives a set of interconnection weights that can be used as a classification network on GLiTCH, after the unknown characters are resampled to the standard 16×32 pixels.

The GLiTCH array implements a network with 512 inputs—which are the binary pixel values in the resampled character—and 32 outputs, corresponding to the 32 character classes to be identified. In an array of 2,048 PEs the 512 weights connected to each output node are stored in 32 groups of 64 PEs. These groups accumulate the sum of the products of the input values and their corresponding weights. Since four times as many PEs as input values exist, four copies of the character to be identified circulate around the array simultaneously, increasing the number of input-weight products calculated in parallel.

The Perceptron weights stored in processor RAM must pass to the CAM for the character classification procedure. All processing for this procedure uses bit-serial arithmetic, and each character requires about 1,200 cycles, of which 27 percent involves inter-PE communication. Substituting the Perceptron for hole counting and template matching yields a total expected time of 542,765 cycles, or 27.1 ms at 20 MHz. The simple Perceptron does not achieve better than about 80 percent successful classification. However, it demonstrated the possibility of using GLiTCH to simulate neural networks.

THE RANGE OF TASKS NEEDED IN A COMPUTER VISION system requires a variety of parallel processing resources. For example, DSP chips can support linear filters, SIMD arrays support nonlinear filters, and MIMD arrays match models. The sequence in which these resources are needed and the relative computer power required of each are not the same for all computer vision tasks. We envisage, therefore, a number of small parallel processing modules that can be combined to provide the appropriate resources for a given vision task.

One such module is based on the GLiTCH associative processor. It provides fine-grain, SIMD parallelism with, typically, 2,048-pixel processors in each module. The bit-parallel, word-parallel associativity available with GLiTCH supports a variety of processing modes—for fast bit-serial arithmetic, as an inverted lookup table, and for bit-parallel arithmetic on sparse arrays. As a demonstration of the effectiveness of

Table 1. Examples of image processing performance for an array of 16 GLiTCH chips (1,024 PEs) with a 20-MHz clock and processing 256×256, 8-bit images.

Operation	Time (ms)
3×3 Laplacian edge enhance	1.4
256-bin intensity histogram	5.0
Hough transform	20.0
Connected component labeling, 400-pixel region	23.0 µs
Resample with bilinear interpolation	3-60
31×31 Laplacian of Gaussian edge detect	90.0
Fast Fourier transform	150.0

GLiTCH, we have shown how a GLiTCH module reads British vehicle number plates at video frame rates. Table 1 summarizes GLiTCH performance for some other image processing operations.

Most of the GLiTCH chip is a regular array of abutting cells. This arrangement lets us use CAD tools to generate, automatically, the silicon layout for GLiTCH and variants of it. The parameters used by the CAD tools can also be used by a compiler generator to produce software tools to match the chip specification.

Acknowledgments

We gratefully acknowledge the contributions by J. Goldfinch (UMIST), I. Hamood (Bristol), A. Brandao, and J.M. Fernandes (Braga, Portugal), and the financial assistance of SERC (Science and Engineering Research Council), BP Research Centre, and DRA (the Defence Research Agency, Electronics Division).

References

1. D.B. Shu, J.G. Nash, and C.C. Weems, "A Multiple-Level Heterogeneous Architecture for Image Understanding," *Application-Specific Array Processors,* S.-Y. Kung et all., eds., IEEE Computer Society Press, Los Alamitos, Calif., pp. 615–627.
2. "MAXbus Specification," Tech. Report SP00-4, Datacube Inc., Dec. 1987.
3. A.W.G. Duller et al., "The Design of an Associative Processor Array," *IEE Proc. Pt. E, Computers and Digital Techniques*, Vol. 136, No. 5, Sept. 1989, pp. 374-382.
4. A.R. Thomson, "A CAM-Based Processor Array for Real-Time Image Applications," PhD thesis, Faculty of Engineering, Bristol University, 1992.
5. R. Storer, "A Content-Addressable Parallel Processor and Its Application to Synthetic Image Generation," PhD thesis, Faculty of Engineering, Bristol University, 1991.
6. A.P. Marriott et al., "Towards the Automated Design of Application Specific Array Processors," *Proc. Int'l Conf. ASAP*, IEEE Computer Society Press, Los Alamitos, Calif., Sept. 1990.
7. R.A. Lotufo et al., "A Transputer-Based Automatic Number-Plate Recognition System," *Proc. Second Int'l Conf. Applications of Transputers*, July 1990, pp. 196-202.
8. P.D. Wasserman, *Neural Computing—Theory and Practice*, Van Nostrand Reinhold, 1989.

Richard Storer is a visiting research fellow at Bristol University where he has been researching parallel processing. He has taught mathematics and computing at several schools and colleges.

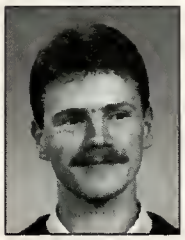
Storer received a BSc in electronic engineering from the University of Wales at Bangor and his PhD from Bristol University for research in massively parallel image rendering. He is a member of the IEEE Computer Society.

Mike R. Pout is a research assistant at Bristol University working on computer vision problems. His technical interests also include benchmarking and parallel processing. He received the BEng degree in computer systems engineering from Bristol University, where he also pursued postgraduate studies.

Andrew R. Thomson is a visiting research fellow at Bristol University. His interests include associative processors, highly parallel computer architectures, and image processing. He received a BEng in computer systems engineering and has been studying for a PhD degree.

Erik L. Dagless is professor of microelectronics and holds a Royal Society Leverhulme Trust Senior Research Fellowship at the University of Bristol. His research interests include the area of digital systems design, specializing in electronic CAD and design methods, associative array architectures, and image and vision processing architectures.

Dagless received his BSc and PhD degrees in electrical and electronic engineering from the University of Surrey. He is a fellow of the Institute of Electrical Engineers, a chartered engineer, honorary editor of Part E of the *Proceedings of the IEE, Computers and Digital Techniques*, and consultant editor for the Addison Wesley Electronic Systems Engineering Series.



Andrew W.G. Duller is a lecturer at the University of Wales, Bangor. His research interests include highly parallel computer architectures, image processing, and neural networks. He spent five years working on associative processor arrays at the University of Bristol where he received a BSc in mathematics and a PhD in computer science.

A. Paul Marriott currently is the invited visiting professor of VLSI design at the Universidade de Aveiro, Portugal, where he leads a team designing a RISC processor. His research interests include massively parallel processors, integrated smart sensor arrays using cellular automata architectures, and automated synthesis of computer architectures.

Marriott received the BSc, MSc, and PhD degrees from the University of Manchester Institute of Science and Technology, England.

Peter J. Hicks is professor of microelectronic circuit design in the Department of Electrical Engineering and Electronics at the University of Manchester Institute of Science and Technology. He developed a masters course in integrated circuit design at UMIST, and managed its IC Design and Test Centre from 1984 to 1990. His research interests include integrated sensor arrays and applications of VLSI in computer vision.

Hicks received the BSc and PhD degrees in physics from the University of Manchester.

Direct any correspondence regarding this article by e-mail to Erik.Dagless@uk.ac.bristol.

Cascading Content-Addressable Memories

We survey the various methods of connecting multiple CAM devices to form a memory system of larger dimensions. The number of elements and the number of data digits per element can be increased relatively easily in contrast to increasing the number of label digits per element, which may be achieved using element, master-slave, or new trie cascades.

Tim Moors

Antonio Cantoni

University of Western Australia

A key factor in the economical use of the "ubiquitous 'random access' memory (RAM)" has been the designer's ability to "cascade" standard mass-produced components. Such cascades form memory systems of variable dimensions (for example, word widths) suitable for different applications. Similarly, combining content-addressable memory (CAM) devices to form a cascade of a larger dimension allows standard CAMs to support different applications without the need for custom devices. Although cascading is almost a trivial matter for RAMs, it is not for CAMs. Therefore, we focus on methods of cascading CAMs. Throughout this article readers may substitute CAMs and RAMs with their read-only counterparts (PLAs and ROMs).

Both CAMs and RAMs select elements of storage when every digit of a supplied "comparand" exactly matches the corresponding digit of the element's explicit or implicit label. We define "comparand" as the value the application supplies to the CAM for comparison. Associative memories, in contrast, select elements by inexact matches, for example, elements with the smallest Hamming distance from the comparand.

A common misconception is that RAMs and CAMs have complementary functionality. For example, when reading, a RAM uses a supplied address to read a value (using an 8-bit "house" number to read a 32-bit name of the "owner" of that number.) And a CAM, on the other hand, uses a supplied value to read an address. However, a RAM cannot distinguish names and house numbers; it just as readily uses a name to index a table of house numbers as it uses house numbers to index a table of names.

The key advantage of CAMs, for most applications, is that they can provide element storage for an arbitrary subset of N_e, possibly nonconsecutive, labels from the label space of 2^l labels. Here, N_e is the number of elements, and l is the number of bits in the comparand. This capability significantly lowers storage requirements when $N_e << 2^l$; that is, when names are used as labels ($l = 32$) rather than house numbers ($l = 8$). We look at other features that may distinguish a CAM from a RAM in the next section.

The added functionality of CAMs comes at a cost: Elements can no longer be identified by their spatial position as is done in a RAM. Instead, each element must include storage for both data and associated label(s), and the CAM must contain comparison logic for comparing these labels with the comparand. The combination of technical factors and lower market demand has limited the capacity of current commercial CAM devices[1-3] and ASIC blocks[4] to around 2^{16} digits, while 2^{22}-bit RAMs are commercially available.

Conventional software-searching algorithms, such as hashing,[5-7] are often inappropriate for hardware implementations because of the vari-

0272-1732/92/0600-0056$03.00 © 1992 IEEE

ance in their processing time. Although their mean delay may be small, the worst-case delay is often much larger, and it is the worst-case delay that is important for synchronous hardware implementations. (Pei and Zukowski[8] examined hardware implementations using tries,[5,7] a common software data structure. Later, we pursue the tries concept in the context of cascading CAMs.) Hence, despite their higher cost, CAMs have found widespread use[5] in such diverse applications as dataflow computers, address filtering for communications networks, and cached memory management.

However, the varied applications have differing requirements of the CAM in terms of its dimension, speed, and functionality. The overview of CAM operations in this article sets the context for a survey of known approaches for combining CAMs to form a cascade whose dimensions differ from that of the constituent devices. One of the methods examined for increasing the label size is a new trie cascade approach.

It is critical to maintain the perspective that RAMs are a specific, constrained type of CAM. RAMs can be used as the constituent "CAMs" in a cascade, provided they satisfy the requirements of a particular cascading method. Correspondingly, CAMs are similar to RAMs, and conventional RAM techniques, such as caching, can be applied to CAMs.

CAM operations

The four primitive operations available in a CAM are selection, multiple response resolution, reads, and writes.

Selection. In general, the comparand and elements of a CAM may use ternary logic, storing information as ternary digits (trits), which may assume the values 0, 1, or X (don't care). Here, the X value matches both 0 and 1. To interface ternary to binary logic, the CAM must use more than one bit to represent each trit. For example, a mask bit may specify whether a corresponding comparand bit should be interpreted as X. By definition, all element digits in the same position as a bit in the comparand form the element's label. The remaining element digits, corresponding to comparand X's, form the data for the element. We generally assume that any string of X's in the comparand is a known length. For example, to select three-character elements containing α anywhere (for example, $\alpha\delta\gamma$ and $\varphi\alpha\delta$), the CAM cannot use a single comparand *α*, where * denotes a variable number of X's. Instead, it must use the three comparands αXX, XαX, and XXα.

The application using the CAM can specify the element digits that are to be interpreted as the label by varying the position, and possibly the number, of X's in the comparand. It may be that some P "pure data" digits of the element cannot be used for labeling, but only for storing data. The comparand digits corresponding to these pure data digits implicitly carry an X value.

RAMs, in contrast, use binary logic for both data and labels. The positions of X values in the comparand are fixed and correspond to pure data digits of the element, with the address forming the remainder of the comparand. Since RAMs provide element storage for each possible comparand (address) ($N_l = N_e = 2^l$, where N_l and N_e are the number of distinct labels and elements in the CAM), a selection operation will select only one element.

We assume each CAM element has a fixed number E of digits, although labels and data may both be of variable size l and d, with $0 \le l \quad E - P$, and $d \equiv E - l$, as shown in Figure 1. Clearly, if either l or d of a CAM is larger than needed for an application, the application can pad its labels and data to match their size to that of the CAM.

Since all CAM comparand digits are treated equally (every digit must exactly match the corresponding label digit), digit ordering is irrelevant, provided the ordering used for the comparand is consistent with that used for the elements. This feature is the same for RAMs, although it contrasts with the more general associative memories. In associative memories, elements are selected if the value of their label is, in some way, associated to the value of a supplied label (for example, the memory selects elements with labels greater than the supplied label). Hence in associative memories, the meaning, and thus ordering, of digits may be important.

The CAM can store the vector indicating which elements have been selected by a previous operation and use this as a state variable for controlling subsequent operations on (un)selected elements. Thus, CAM selection operations may be cascaded. For example, elements may be selected when they match the current comparand and the previous operation had selected an adjacent element.

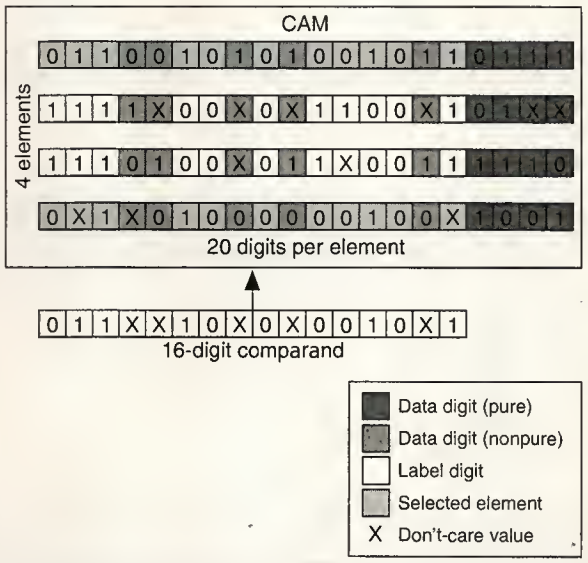


Figure 1. CAM parameters with values $N_e = 4$, $N_l = 2$, $P = 4$, $d = 9$, $l = 11$, and $E = 20$.

Multiple response resolution. After the selection operation(s), multiple elements may have been selected either because labels were not unique or because selections were Ored. RAMs differ in that only one element can be selected at any time. If the operation following selection is an element-serial one (reading data from a particular selected element or writing to *an* unused element), the CAM must resolve the multiple responses of the selection operation to establish the order of the processing. Since the CAM selects elements by exact matches to the comparand, no restriction exists on the order in which multiple response resolution must choose selected elements for serial processing.

Reads. The read operation returns an indicator of the number of selected elements (none, one, or multiple) and part or all of the information affiliated with these elements. Since CAMs generally provide a data bus capable of carrying information from only one element, the CAM must resolve the multiple responses prior to the element-serial reading of selected elements.

Writes. Write operations modify the label or data of all selected elements using supplied parameters that specify which digits to modify and how to modify them. The ability to selectively mask which data may be modified can be useful[9] (for example, when simulating multiple CAMs by time-multiplexing a single CAM). Selective masking avoids a read-modify-write sequence, which would also serialize the modification. The value to which a digit is modified may be an arbitrary function of the existing and supplied digits (Nand digits), although usually the supplied digit overwrites the existing digit.

Since multiple elements may be selected for a write operation, CAMs, in contrast to RAMs, may concurrently write common information to multiple elements in parallel. This powerful feature aids some applications, such as initializing memory with a test pattern.[10]

Operation application. As an example of the application of these primitive operations, consider Figure 2 showing a CAM used for cache memory management. Each element of the CAM contains storage for a cached block's main memory address, its attributes indicating how recently it has been accessed, and its cache address. To access the memory hierarchy, the processor supplies a main memory address as a comparand to the CAM, which selects any element describing a cached block with a matching main memory address.

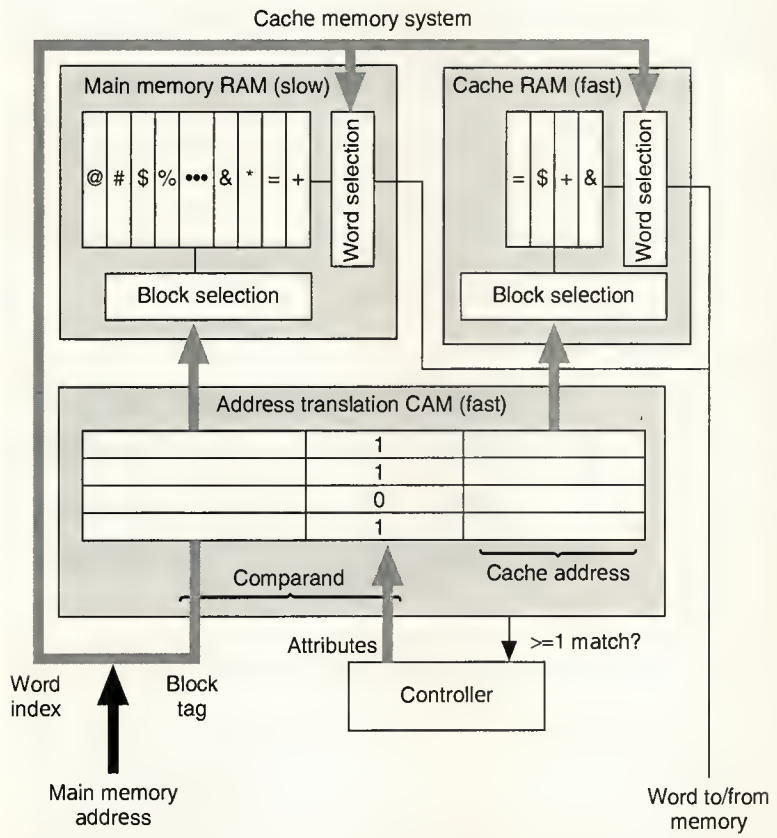


Figure 2. Application of a CAM to cached memory management. The cache address may be pure data and is often statically encoded. The symbols represent values of blocks of memory in the main memory and cache.

A read operation then determines whether a matching label exists (a cache hit). If so, the cache controller reads the cache address for this block and uses it to address the cache memory. It overwrites the single-bit attributes field to record that the block has recently been accessed. If the selection failed to find a match for the main memory address, the cache controller must select a cache block for replacement using the attributes as a label. For example, it may select the least recently used block. (Designers place the block in the cache in anticipation that the same, or other, address(es) in the block will be accessed in the near future.)

The change in which element digits are to be interpreted as the label requires that the comparand mask be changed to indicate that the main memory block address bits are X. The cache controller then reads the cache address for this replacement block to control the transfer of the block from main memory to the cache and updates the main memory address data. If multiple blocks are equally suited to replacement (accessed at the same time), the CAM must resolve the multiple responses.

The fast component of this memory system

can be seen as being intermediate between a pure CAM and a pure RAM. Although storage is not provided for all word labels in the global label space, the fast component does provide storage for all labels in a number of local label spaces corresponding to blocks in the cache. Thus, it is evident that CAMs and RAMs form two extremes of a spectrum of memory systems that includes memories such as the fast component of this memory system.

In this cache application the CAM elements will continually be modified as blocks transfer to and from the cache. It is necessary to maximize the throughput of CAM operations to match the rate at which the processor cycles through new addresses and desirable to minimize the delay of CAM operations to minimize the branch penalty.

For other applications the demands of the CAM may be different. For example, consider a CCITT Broadband ISDN switch[11] in which a CAM is used to determine the output port for an incoming cell of information via the cell's label. The switch may have relatively static CAM elements corresponding to connections through the switch. It may require high throughput (over a million searches per second for 620-Mbps links), although delays significantly larger than the cycle time (for example, milliseconds) may be considered acceptable. The dimensions of the CAM would vary between implementations. That is, the number of elements corresponding to the number of concurrent connections supported, the number of data digits per element to the number of output ports, and the number of label digits per element would depend on which components of the cell label are used for switching (for example, virtual path or channel identifiers).

Thus, the varied applications of CAMs have differing requirements of the CAM in terms of its dimension, speed, and cost. Different cascading methods may be better suited to different applications.

CAM realization

Though modern VLSI (very large scale integration) processes permit the integration of millions of devices on one chip, interchip connectivity remains a bottleneck. Limited bonding pads and packaging costs restrict connectivity to, at most, hundreds of connections of limited throughput. Indeed, packaging dominates the cost of many chips. For a given storage capacity of N_l elements with different labels, a CAM providing $N_l < 2^l$ will require larger labels l than an equivalent RAM ($N_l = 2^l$). Additional pins, leading to increased costs, can be avoided by time-multiplexing the pins, at the expense of the time it takes to transfer information. Often, some externally supplied information is essentially invariant (the comparand masks used in the cache memory example). Therefore, CAM chips can overcome the bottleneck created from the multiplexed inputs by providing registers internal to the CAM chip (a cache) for storing this information, as is done in the GEC Plessey PNC1480.[3]

Cascading CAMs

We can increase the size of a CAM in any of its dimensions: the number of elements N_e, the number of data digits per element d, or the number of label digits per element l.

We can compare the methods for cascading CAMs in terms of their cost, functionality, and speed. We can assess these parameters for the constituent CAMs, the interconnections between constituent CAMs, and for the cascade itself.

We assess the cost of the CAMs in terms of the number of storage cells required and the cell's type (pure data or label). The cost of the connectivity is based on the number of inter-CAM connections required.

Functionality covers such issues as the ability to use ternary rather than binary logic and the potential for parallel writing and for reading an indication of the number of selected elements. Cascade functionality also encompasses maintenance of the cascade: how elements are added/removed, how labels and data can be modified, and whether such modification interferes with other concurrent operations.

Speed concerns the measurement of both the delay t_d for an operation to be performed and the cycle time t_c between operations. Although different operations may have different speeds, we use the selection operation as a benchmark. It is a reasonable choice since the speeds of other operations are mostly invariant between different methods of cascading to extend the CAM in a given dimension.

We can use concurrency to reduce t_c below t_d by either replicating the memory or by pipelining memory stages. Pipelining is often the preferred technique since the hardware overheads are often lower and the problem of consistency between memories is reduced. Realization factors—such as the necessity to refresh dynamic storage after a destructive readout and overheads between pipeline stages—ultimately impose a lower limit on t_c that is achievable through pipelining. In applications exhibiting strong spatial locality, we can divide elements with adjacent labels between replicated memories rather than duplicating them in each memory. By doing this, we achieve improved memory utilization and throughput. This method is well known in the context of RAMs as interleaving.

Adding elements

To increase the number of elements, CAMs may share a common comparand bus, as shown in Figure 3. For selection, CAMs do not need to interact since the selection of an element is independent of that of other elements. Hence, selection within a CAM remains independent of that of other CAMs.

Complexities arise in the subsequent processing of selected elements. To read an indicator of the number of selected elements in the cascade, the cascade must have added the number selected in each CAM. For example, a simple indicator of the range of the number of selected elements is whether

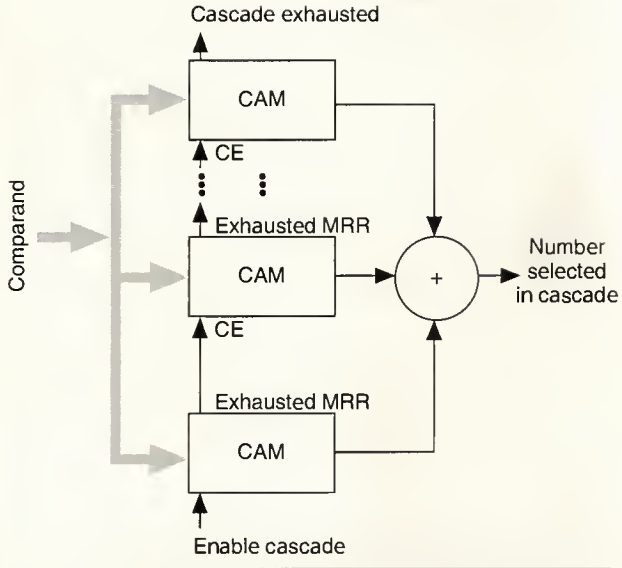


Figure 3. Increasing the number of elements. Element-serial operations require daisy-chaining connections.

at least one match for the comparand exists in the cascade. To determine if there is at least one match, the CAM must Or the match signals from the constituent CAMs.

Multiple response resolution, to be performed prior to serial operations, must account for selected elements in all CAMs of the cascade. Since multiple response resolution may be performed in arbitrary order for selected CAM elements, designers typically achieve the ordering by daisy-chaining CAMs.[12] As shown in Figure 3, a chip enable signal CE ripples through the cascade: A CAM enables the next CAM in the chain only if it has exhausted all of its selected elements. Carry-lookahead logic can increase the propagation speed of the enable signal.[13,14]

Although static element identifiers may be locally unique within each CAM, making them globally unique within the cascade requires that a unique identifier of the CAM in which the element is stored be added to the locally unique identifier.

Increasing data size

We may also want to cascade CAMs to increase the number of data digits per element because we have too few data digits per element, or because the data digits that are available are read-only. For example, many CAMs[1,3] statically assign each element a unique identifier (data) that must be used in a cascade to access a read/write memory.

We can use the same brute-force approach used in RAMs to increase the number of element digits available for data storage. Specifically, we can replicate labels for each element in multiple CAMs and distribute the data for each element among the CAMs, as shown in Figure 4a. Clearly, this approach provides a cascade element data capacity of pd, where p is the number of parallel CAMs. Such additional storage can only be used for pure data; it cannot later be used for labeling elements when the comparand changes. Furthermore, we cannot use this approach to provide writable storage when the CAMs have read-only data.

If CAM elements are assigned unique identifiers (statically or using $\lceil \log_2(N_e) \rceil$ bits of the available data storage), we can use these identifiers to index another CAM/RAM, which provides additional data storage as shown in Figure 4b. Although the functions of element identification and of data element selection using this identifier effectively cancel each other, they do introduce a serial bottleneck that prevents parallel writing to the supplementary data storage. Again, the additional storage is for pure data only.

A third approach to increasing the number of digits avail-

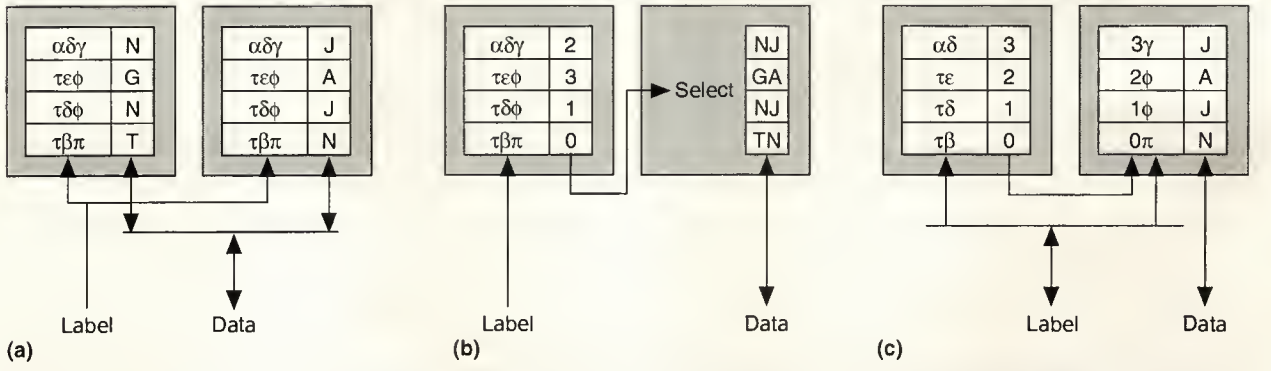


Figure 4. Cascading to increase data storage: replicating labels (a), identifiers indexing second RAM/CAM (b), and reducing label redundancy (c). Letters indicate the data associated with the labels.

able for data storage is to reduce the number of digits occupied by the label. Say $l > \log_b(N_l)$, where N_l N_e is the number of distinct labels in the CAM, and b is the base of the label digits (for example, $b = 2$ for binary). Then, a redundancy exists in the labels, which the additional CAM stage shown in Figure 4c can remove, and in doing so, increase the space available in the main CAM for data storage. This result comes at the expense of the additional CAM and increasing the selection delay to the sum of the delays in the constituent CAMs.

Increasing label size

In examining cascading to increase the label size, we assume that we seek cascade labels individually larger than the labels of constituent CAM chips ($\exists\, l_i : l_i > E - P$). We do not seek the combination of labels for an element to be larger than the space available in the CAM chips ($\Sigma\, l_i > E - P$ and $l_i < E - P$ $\forall\, l_i$). If the latter is true, we can use CAM chips in parallel without interaction, with as many labels per CAM chip as will fit. This structure is limited in that only the component of the element that was stored in the CAM matching the comparand will be selected. Other components, for example, other labels, will not be (directly) selected.

When labels are individually larger than that supported by the constituent CAMs, we must divide the comparand (and labels) into segments that are distributed between logically distinct CAMs. Some interaction must exist between these CAMs since it is not sufficient that each CAM finds a match for its segment of the comparand. The CAMs must match a common cascade element. For example, when a cascade of three CAMs stores the labels αδγ, τεφ, τδφ, and τβπ, with one character of each label per CAM, it should find no match for αδφ. However, matches would exist for each of these characters in the appropriate CAMs.

Before examining methods for cascading CAMs to raise the label size to that required by an application, we should examine how an application requiring smaller labels can use a CAM with larger labels. We need to examine this since it may be possible to manufacture CAMs with large labels and for applications to merge multiple smaller labels into the larger CAM label storage. As mentioned earlier, we can pad labels to match their size with that of a CAM; however we will end up poorly utilizing each CAM element.

By using ternary comparands, a CAM can simulate multiple CAMs of total label size $\Sigma\, l_i$ $E - P$, and total data size Σd_i $E - \Sigma\, l_i$. The CAM divides the E digits of the large elements between the smaller elements ($l_i + d_i$) and time-multiplexes the CAM. Large-element digits corresponding to other labels are masked out with an X in the comparand, as shown in Figure 5. From this, it is apparent that CAMs could be manufactured with extremely large elements, and applications requiring smaller elements could time-multiplex the large elements. However,

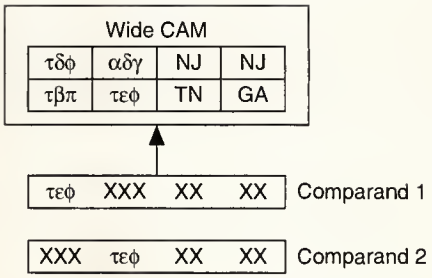


Figure 5. Using a wide CAM for thinner labels through time-multiplexing.

- The limited interchip connectivity would form a bottleneck for transferring large comparands to the CAM.
- As thinner elements are used, the CAM elements must be multiplexed more often for each operation, mitigating the speed advantage of CAMs resulting from operations being performed concurrently on all elements.
- To provide for modification of a subelement of a shared element, we require either maskable writing, or read-modify-write sequences that will degrade performance.
- Irrespective of the capacity of a single CAM chip, some applications may exist whose labels are larger than the total storage provided in a single CAM.

Thus, we seek methods for cascading thin CAMs to form a wide cascade. Since each method provides different cost, functionality, and speed, they suit different applications.

Element cascading. Since exact matches between the comparand and element are required in a CAM, the comparison between the comparand and element will produce a binary result. One approach for cascading CAMs is to divide the segments of the comparand into logically distinct CAMs, and to And the results from matching each segment of the comparand and label for each element. The cascade selects an element only if all segments match the corresponding comparand segments. This "element cascade" approach, shown in Figure 6a, requires at least one inter-CAM connection per element for Anding the match results. Thus, the inter-CAM connectivity will be high for even a moderate N_e ($N_e = 256$), and this approach is suitable not for cascading discrete CAM devices but rather for providing multidigit labels within a CAM device.

Rather than use physically distinct CAM chips, a single CAM chip can be time-multiplexed to emulate multiple CAM chips. This approach exploits the massive connectivity available on chip to provide the intraelement connectivity. In this approach,[2,13,15] we store segments of a cascade element in adjacent elements of the CAM. For example, segment s_i of Figure 6a is positioned below and adjacent to s_{i+1} as shown in Figure 6b. The CAM uses the first segment of the cascade comparand

to select cascade elements with matching first segments. Subsequent selection operations for other cascade segments select an element if it matches the comparand and its lower neighbor element matched in the previous selection operation.

To enable a CAM chip to be used with arbitrary length comparands, it should provide connectivity between all adjacent elements for the propagation of match results. We can do this by using a shift register, as shown by the screened connections of Figure 6b. The only required inter-CAM connectivity is two connections per CAM device to concatenate its shift register with those of adjacent devices. However, it is not sufficient for successive elements of the CAM to match the successive segments of the cascade comparand. These elements must also be in the same cascade element and not split across cascade elements. For example, in Figure 6b, no match should be found for φαδ, even though these characters are stored in consecutive cascade elements storing τεφ and αδγ.

We can ensure the consecutive matches of cascade comparand segments occur within a common cascade element in a number of ways, including the following:

- *A maskable decoder/selector[13] can be used to limit the set of candidates for the first selection operation to the first element of each cascade element.* Often the decoder can only be masked to select CAM elements a power-of-2 elements apart, and thus cascade element lengths would be limited to CAM element lengths multiplied by a power of 2. This restriction on element lengths will affect the efficiency of memory utilization.
- *One or more of the segments of the comparand and elements can be anchored together,[16] ensuring that the cascade comparand segment only matches the corresponding segment of the cascade element.* We can implement this step in two ways. We can insert unique spacer code words that match only spacer words inserted in the comparand and mismatch the comparand segments. Or we can tag each segment of the element[15] and append the appropriate tag to each comparand segment.

Since only one anchor is required per cascade element, we can use a single-bit delimiter for tagging.[17] Not only does this minimize the per-

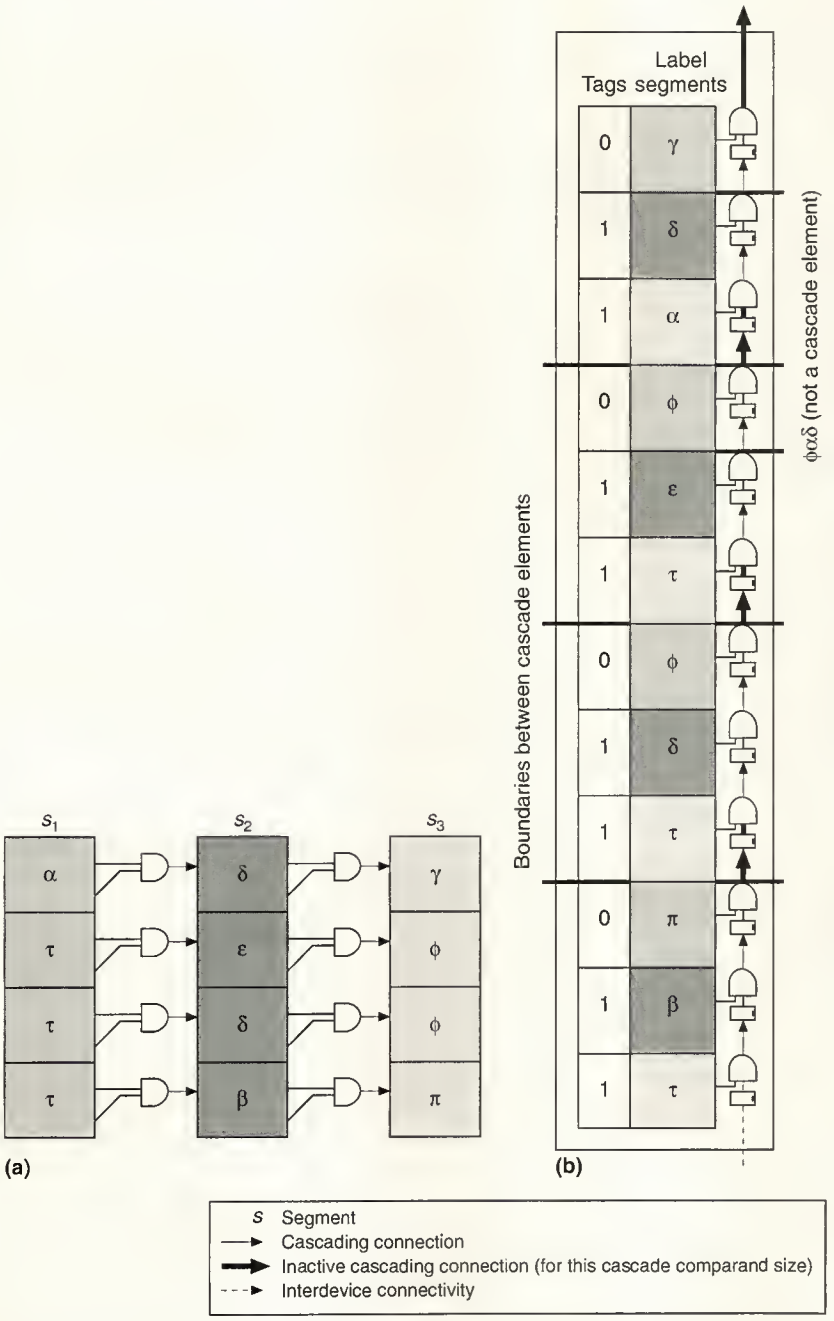


Figure 6. Element cascade: logical representation, for example, physically distinct CAMs (a) and within a single physical CAM, (b).

element overhead for storing tags, but it also obviates the need to supply tags as part of the comparand. The delimiter bit can enable the shift register (delimiter bit set to True for last segment of cascade element). Furthermore, by using a single-bit tag, we can use comparands that have a variable-length string of X's before the binary digits in the comparand. Thus, we detect any string of consecutive segments in the CAM matching the string of comparand segments. After selecting the first binary comparand segment, the CAM selects elements if their lower neighbor was selected in the previous operation (don't-care comparand). Provided cascade elements are of uniform size, repeating this don't-care selection $\lceil l_C / l \rceil - \lceil c_C / c \rceil$ times ensures that a match signal will propagate to the last segment only when the string of segments was wholly within the one cascade element. (Parameters with a subscripted C suffix refer to the cascade of CAMs rather than to a single CAM.)

In any of its forms, this element cascade approach permits virtually unlimited increases in the cascade label size in increments of the CAM label size. Yet, it requires only two inter-CAM connections per CAM device. As the number of required devices is proportional to $\lceil l_C/l \rceil N_{e,C}$, we can trade increases in the element size for the number of elements, without the need for additional devices. Furthermore, this approach does not require either elements or comparands to be binary rather than ternary. The cost comes in the form of either the anchoring or the maskable decoder, and through a reduction in speed. Both t_d and t_c increase in proportion to $\lceil l_C/l \rceil$. Unfortunately, some CAMs[1,3] do not provide the required shift register for cascading selection operations. Furthermore, for some applications, high throughput is of paramount importance. Hence, we investigate other cascading methods.

Master-slave cascading. To reduce the inter-CAM connectivity from that of the element cascade without a shift register, each CAM can pass a unique identifier of candidate element(s) (of at least $\log_2(N_e)$ bits to ensure uniqueness) to the other CAMs. When there are multiple candidates, the CAM must pass the identifiers serially, degrading performance.

One approach, shown in Figure 7a, assigns a unique identifier to each element in the cascade.[18]

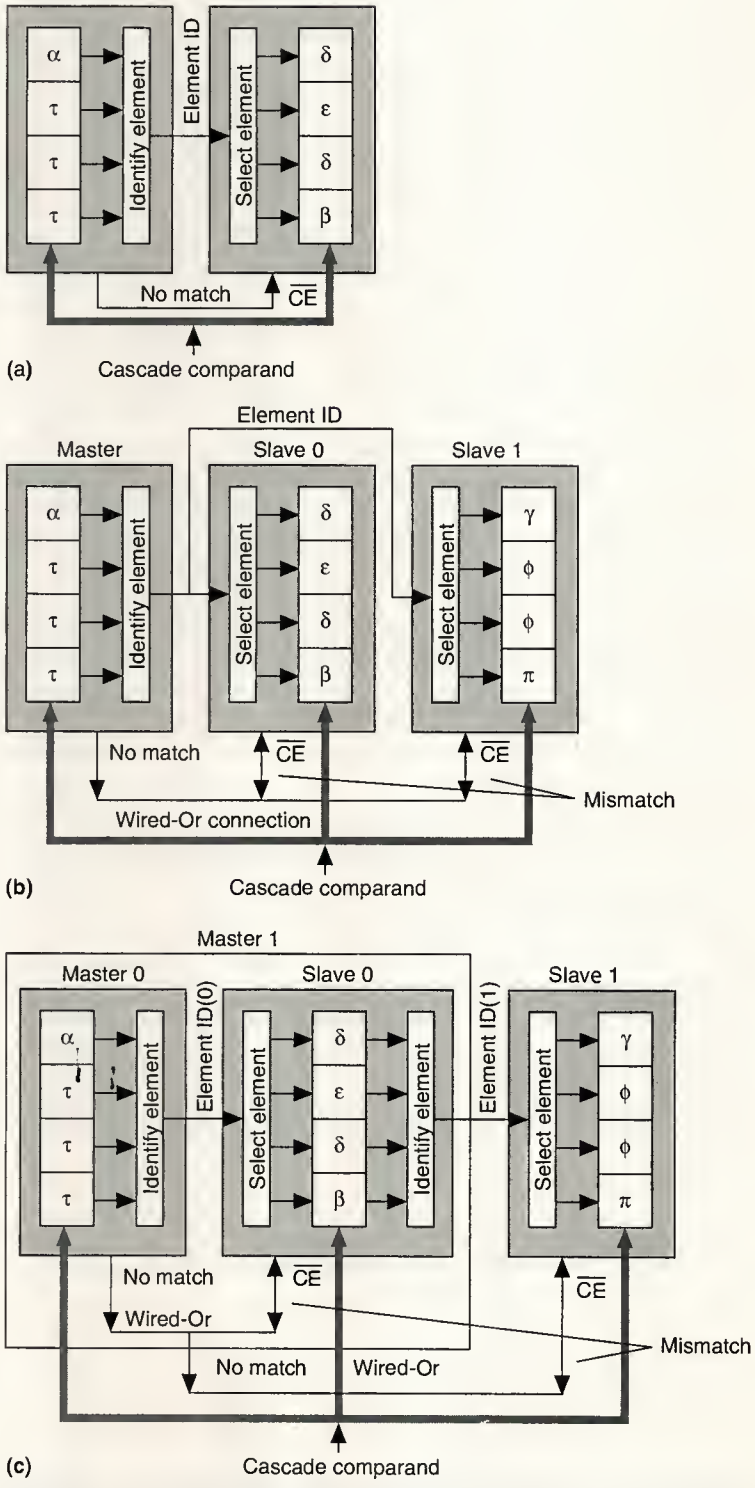


Figure 7. Master-slave cascading: one master and slave (a), master and multiple slaves (b), and multiple master-slave pairs (c).

A master CAM locates cascade elements that match the comparand in the master segment, and it passes the identities of these candidate elements to the slave CAM(s). Each of the slaves then attempts to match its comparand segment with the element identified as a candidate by the master. Only if all slaves match, as determined by Anding the match signals for each CAM chip, does the cascade select the element.

The master-slave approach suffers from the serial transfer of element identities from the master to slave(s). The worst case occurs when the master is full of labels matching the master's segment of the cascade comparand. This approach, then, takes a time proportional to N_e to select all matching elements or to ensure there are no matches. Even when N_e is only moderate, for example, $N_e = 256$, the worst-case delay will be extensive. Since we use CAMs rather than approaches such as hashing to avoid delay variance, the master-slave approach is not well suited to cascading CAMs. Furthermore, the slaves work in an element-serial fashion: They need only compare their segment of the cascade comparand to one label (as specified by the master's identifier) at any instant. Thus, any potential the slave CAMs may have for concurrent element comparisons (using distributed logic) is wasted. The slaves may as well be RAM lookup tables indexed by the identifier from the master, with the indexed entry compared to the segment of the comparand, as implemented in cache address comparator chips.[19]

We can extend the master-slave structure by using multiple slaves for the one master, as shown in Figure 7b. Alternatively, we can use a master-slave cascade as either the master or slave of a new cascade, as shown in Figure 7c. The latter approach would not be used for cascading since the selection delay would increase in proportion to the number of CAMs in the cascade (compared to the single-master approach in which the delay through the master and the slowest of the slaves). However, it forms a structure that can be used for trie cascading.

Trie cascading. Clearly, we can avoid the master-slave serialization by merging the CAM_1 (master) label storage for elements with common CAM_1 comparand segments. As multiple cascade elements can now share a common CAM_1 element, it is no longer possible to pass a cascade element identifier between CAMs and use the structure of Figure 7b. Rather, we must use the structure shown in Figure 7c, and the identity of the selected element in CAM_i is combined with segment $i + 1$ of the comparand to form the comparand for CAM_{i+1}, as shown in Figure 8.

We can understand this trie cascading approach[20] by representing the search space as a tree, as shown in Figure 9. Here, each path from the root node of the tree to a leaf node corresponds to a label stored in the tree, with data stored in the leaf nodes. Each CAM of the cascade corresponds to a level of the tree, and each occupied CAM element to a branch from that level. As the search progresses from level (CAM) i to the next, the search space reduces to the set of labels that have matched segments 1 to i of the comparand. This is equivalent to the searching algorithm used with trie data structures in software,[5,8,21] hence the name trie cascading. At the extreme, where each CAM comparand includes only a single bit of the cascade comparand, a binary tree forms, and the searching technique is equivalent to that used by Wolstenholme.[22]

Whenever more than one element is selected in a CAM, the identities must be passed serially between CAMs, which will result in the same performance degradation encountered in the master-slave approach. As a result, the labels and comparand can contain X digits in segment i only when all digits in subsequent segments are X's. This corresponds to terminating the search after reaching level i of the tree. Although the cascade elements matching the comparand will not have been selected (not all segments have been processed), this approach provides an indication of a match for the comparand within the cascade.

The identifiers from CAM_i divide the storage in CAM_{i+1} into logically distinct search spaces corresponding to branches from level i. By assigning each branch within the tree a unique identifier, we can store all branch-identifier/element-segment pairs within a single CAM. The advantage of distributing branch storage for different levels in different CAMs is that it enables

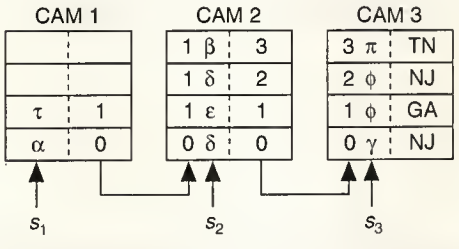


Figure 8. A serial trie cascade with s_x indicating segment x of the comparand.

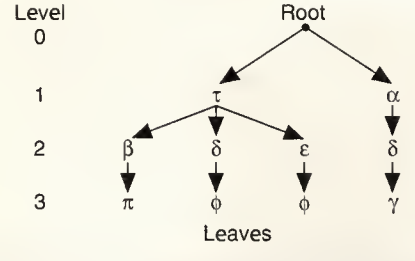


Figure 9. Tree representation of labels τβπ, τδφ, τεπ, and αδγ.

READER SERVICE CARD

IEEE MICRO

June 1992 issue (card void after December 1992)

Name _____

Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Country _____ Phone(_____) _____

Reader Interest

(Add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles and departments):

150	151	152	180	181	182
153	154	155	183	184	185
156	157	158	186	187	188
159	160	161	189	190	191
162	163	164	192	193	194
165	166	167	195	196	197
168	169	170	198	199	200
171	172	173	201	202	203
174	175	178	204	205	206
177	178	179	207	208	209

Product Information

(Circle the numbers to receive product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	
11	31	51	71	91	111	131	
12	32	52	72	92	112	132	
13	33	53	73	93	113	133	
14	34	54	74	94	114	134	
15	35	55	75	95	115	135	
16	36	56	76	96	116	136	
17	37	57	77	97	117	137	
18	38	58	78	98	118	138	
19	39	59	79	99	119	139	
20	40	60	80	100	120	140	

READER SERVICE CARD

IEEE MICRO

June 1992 issue (card void after December 1992)

Name _____

Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Country _____ Phone(_____) _____

Reader Interest

(Add comments on the back)

Readers,
Indicate your interest in articles and departments **by circling the appropriate number** (shown on the last page of articles and departments):

150	151	152	180	181	182
153	154	155	183	184	185
156	157	158	186	187	188
159	160	161	189	190	191
162	163	164	192	193	194
165	166	167	195	196	197
168	169	170	198	199	200
171	172	173	201	202	203
174	175	176	204	205	206
177	178	179	207	208	209

Product Information

(Circle the numbers to receive product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	
11	31	51	71	91	111	131	
12	32	52	72	92	112	132	
13	33	53	73	93	113	133	
14	34	54	74	94	114	134	
15	35	55	75	95	115	135	
16	36	56	76	96	116	136	
17	37	57	77	97	117	137	
18	38	58	78	98	118	138	
19	39	59	79	99	119	139	
20	40	60	80	100	120	140	

SUBSCRIPTION CARD

SUBSCRIBE TO IEEE MICRO

All the facts about today's chips and systems

❑ **YES**, sign me up!

If you are a member of the Computer Society or any other IEEE society, pay the member rate of only $23 for a year's subscription (six issues).

Society: _____

IEEE membership no: _____

Society members: Subscriptions are annualized. For orders submitted March through August, pay half the full-year rate ($11.50) for three bimonthly issues.

❑ **YES**, sign me up!

If you are a member of ACM, ACS, BCS, IEE (UK), IEEE but not a member of an IEEE society), IECEJ, IPSJ, NSPE, SCS, or other professional society, pay the sister-society rate of only $39 for a year's subscription (six issues).

Organization: _____ Membership no: _____

For airmail option, see page 2.

_____
Full Signature Date

Name _____

Street _____

City _____

State/Country ZIP/Postal Code

❑ Payment enclosed *Residents of CA, DC, Canada, and Belgium add applicable sales tax.*

❑ Charge to ❑ Visa ❑ MasterCard ❑ American Express

Charge-card number

_____

Expiration date

____ ____
Month *Year*

Prices valid through 12/31/92

0692 MICRO

Charge orders also taken by phone:
(714) 821-8380 8 a.m. to 5 p.m. Pacific time
Circulation Dept.
10662 Los Vaqueros Circ., PO Box 3014
Los Alamitos, CA 90720-1264

Editorial comments

I liked: _____

_____

_____

I disliked: _____

_____

_____

I would like to see: _____

_____

_____

Reviewers Needed. *If interested, send professional data to Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129, Torino, Italy.*

For reader service inquiries, see other side.

PLACE POSTAGE HERE

PO Box is for reader service cards only.

IEEE Micro
PO BOX 16508
NORTH HOLLYWOOD CA 91615-6508
USA

Editorial comments

I liked: _____

_____

_____

I disliked: _____

_____

_____

I would like to see: _____

_____

_____

Reviewers Needed. *If interested, send professional data to Dante Del Corso, Dipartimento di Elettronica, Politecnico di Torino, C.so Duca degli Abruzzi, 24, 10129, Torino, Italy.*

For reader service inquiries, see other side.

PLACE POSTAGE HERE

PO Box is for reader service cards only.

IEEE Micro
PO BOX 16508
NORTH HOLLYWOOD CA 91615-6508
USA

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 38 LOS ALAMITOS, CA

POSTAGE WILL BE PAID BY ADDRESSEE

IEEE COMPUTER SOCIETY
PO BOX 3014
LOS ALAMITOS CA 90720-9804
USA

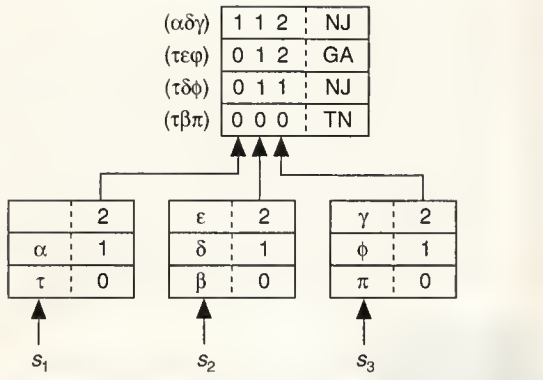


Figure 10. Parallel trie cascade.

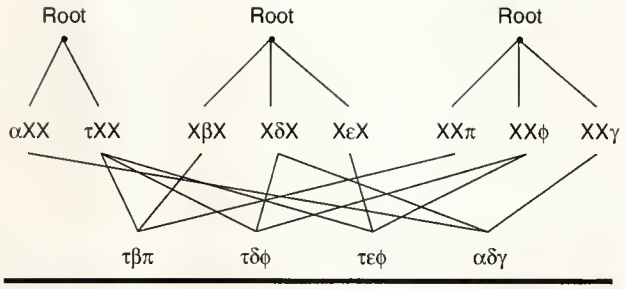


Figure 11. Forest representation of search space.

pipelining of CAM operations. A selection operation that has reached level i does not interfere with one at another level of the tree. Thus, the throughput of CAM operations can remain invariant as label sizes increase. With this serial trie cascade, the delay for CAM operations will increase in proportion to the number of levels in the searching process (label size).

We can reduce the delay for element selection by using multiple CAMs in parallel to concurrently reduce the search space, forming a parallel trie cascade as shown in Figure 10. As multiple CAMs at level i concurrently reduce the search space for level $i + 1$, the delay scales in proportion to the logarithm of the label size, although the throughput remains independent of the label size. This reduction in delay comes at the expense of additional hardware. In terms of the search space, this parallel trie cascade corresponds to a forest structure shown in Figure 11. Here, each root corresponds to a segment of the label, and leaf nodes correspond to target labels in the search space. Searching progresses concurrently from the roots of each of the trees until the search processes converge at a common leaf node.

The trie cascade approach requires inter-CAM connectivity proportional to the logarithm of the comparand size. The throughput is independent of the comparand size, and the delay can increase in proportion to the comparand size or to its logarithm, depending on the implementation. One weakness is that comparands and labels are effectively prevented from containing X's.

Maintenance of the trie cascade, that is, the addition and removal of elements, is also more complicated than for other cascading methods. When adding elements, we must search from the root level for the first level that does not have a branch matching the segment of the label. From this level onward, we must add a branch for each segment of the label. The only requirement of branch identifiers is that they be unique within a CAM. Hence, we can set them at CAM initialization and not have to modify them when elements are added or removed. To remove an element, we must search from the root level for the first level at which the branch used by the label is not shared with another label. This branch, and all branches for the label in subsequent levels, should then be removed. An implementation of element removal may use the CAMs to check for multiple matches of the identifier from CAM_i in CAM_{i+1} and prune the branches from CAM_i onward.

A NUMBER OF APPROACHES EXIST FOR CASCADING CAMs. We can daisy-chain CAMs to increase the number of elements and possibly use carry-lookahead logic to increase the speed at which the enable signal propagates through the cascade. To the data size, we can replicate the labels in distinct CAMs, use the data storage available in a primary CAM to index a secondary CAM/RAM, or reduce redundancy in labels.

To increase the label size, we can use an element cascade with or without a shift register, a master-slave cascade, or a trie cascade. The element cascade without a shift register requires one inter-CAM connection per element; with a shift register it requires two inter-CAM connections per device. With the shift register, the selection delay increases in proportion to the comparand size.

The master-slave cascade requires inter-CAM connectivity proportional to the logarithm of the number of elements, and the worst-case selection delay increases in proportion to the number of elements.

The trie-cascade method also requires inter-CAM connectivity proportional to the logarithm of the number of elements. It provides a throughput independent of the comparand size, and a delay proportional to the comparand size or to its logarithm, depending on the implementation.

Acknowledgments

Grants from the Australian Telecommunications and Electronics Research Board and the Department of Industry, Technology and Commerce, Industry Research and Development Board supported this work.

References

1. "Am99C10 256×48 Content Addressable Memory," *AMD Memory Products*, Advanced Micro Devices, Sunnyvale, Calif., 1989, pp. 4.119-4.144.
2. "CRC32256 and Coherent Processor Product Announcements," Coherent Research Inc., East Syracuse, N.Y., 1991.
3. "PNC1480 64-Kbit Content Addressable Memory: Advance Information," GEC Plessey Semiconductors, Swindon, Wiltshire, UK, June 1991.
4. "LEA100K Embedded Array Series Product Overview," LSI Logic, Milpitas, Calif., May 1991.
5. T. Kohonen, *Content-Addressable Memories*, 2nd ed., Springer-Verlag, Berlin, 1987.
6. D. Knuth, *The Art of Computer Programming, Sorting and Searching*, Vol. 3, Addison-Wesley, Reading, Mass., 1968.
7. E. Fox et al., "Practical Minimal Perfect Hash Functions for Large Databases," *Comm. ACM*, Vol. 35, No. 1, Jan. 1992, pp. 105-121.
8. T. Pei and C. Zukowski, "VLSI Implementation of Routing Tables: Tries and CAMs," *Proc. Infocom*, IEEE Computer Society Press, Los Alamitos, Calif., 1991, pp. 515-524.
9. F. Hermann et al., "A Dynamic Three-State Memory Cell for High-Density Associative Processors," *IEEE J. Solid-State Circuits*, Vol. 26, No. 4, Apr. 1991, pp. 537-541.
10. G. Giles and C. Hunter, "A Methodology for Testing Content Addressable Memories," *Proc. Int'l Test Conf.*, CS Press, 1985, pp. 471-474.
11. "Switching for Broadband Telecommunications," *IEEE J. on Sel. Areas Comm.*, Vol. 5, No. 8, Oct. 1987, pp. 1217-1376.
12. T. Ogura et al., "A 4-Kbit Associative Memory LSI," *IEEE J. Solid-State Circuits*, Vol. 20, No. 6, Dec. 1985, pp. 1277-1281.
13. J. Wade and C. Sodini, "A Ternary Content Addressable Search Engine," *IEEE J. Solid-State Circuits*, Vol. 24, No. 4, Aug. 1989, pp. 1003-1013.
14. A. McAuley and C. Cotton, "A Self-Testing Reconfigurable CAM," *IEEE J. Solid-State Circuits*, Vol. 26, No. 3, Mar. 1991, pp. 257-261.
15. T. Ogura et al., "A 20-Kbit Associative Memory LSI for Artificial Intelligence Machines," *IEEE J. Solid-State Circuits*, Vol. 24, No. 4, Aug. 1989, pp. 1014-1020.
16. K. Takahashi et al., "A New String Search Hardware Architecture for VLSI," *Proc. 13th Ann. Int'l Symp. Computer Architecture*, CS Press, June 1986, pp. 20-27.
17. M. Hirata et al., "A Versatile Data String-Search VLSI," *IEEE J. Solid-State Circuits*, Vol. 23, No. 2, Apr. 1988, pp. 329-335.
18. T. Nikaido et al., "A 1-Kbit Associative Memory LSI," *Jap. J. Appl. Phys.*, Vol. 22, Supp. 22-1, 1983, pp. 51-54.
19. *Cache Memory Management Data Book*, Texas Instruments, Dallas, Tex., 1990.
20. T. Moors et al., "Cascading CAMs for Increased Label Size," Univ. Western Australia Networking Research Laboratory report NRL-TR-3, 1991.
21. E. Fredkin, "Trie Memory," *Comm. ACM*, Vol. 3, No. 9, Sep. 1960, pp. 490-499.
22. P. Wolstenholme, "Filtering of Network Addresses in Real Time by Sequential Decoding," *IEE Proc.*, Part E, Vol. 135, No. 1, Institute of Electrical Engineers, UK, Jan. 1988, pp. 55-59.

Tim Moors is a PhD student in electronics engineering at the University of Western Australia, where he is investigating the interconnection of metropolitan area networks. His technical interests include computer architecture and networking. Moors received a BE degree in electronics engineering from UWA and is a student member of the Institute of Electrical and Electronics Engineers Computer and Communications societies.

Antonio Cantoni is an associate professor in the Department of Electrical and Electronics Engineering at UWA. Earlier, he was director of the Digital and Computer Systems Design Section of QPSX Communications Ltd. in Perth for the development of the DQDB metropolitan area network. He has also been a lecturer in computer science at Australian National University in Canberra and chair of computer engineering at the University of Newcastle in Shortland, New South Wales, Australia.

Cantoni received his BE and PhD degrees in electronics engineering from the University of Western Australia, Nedlands. He is a senior member of the IEEE and a member of the Association of Computing Machinery.

Direct questions concerning this article to Tim Moors, Networking Research Laboratory, Dept. of Electrical and Electronics Engineering, University of Western Australia, Nedlands, WA 6009, Australia; or e-mail at tim@swanee.ee.uwa.oz.au.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

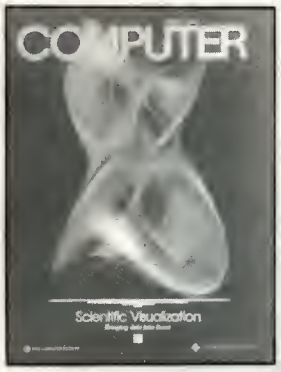
Low 168 Medium 169 High 170

IEEE COMPUTER SOCIETY

1992 Membership/Subscription Application

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

BENEFITS



Receive *Computer* automatically with membership. Written and refereed by experts, it features articles on the latest developments in computer technology, applications, and research, as well as survey and tutorial articles on topics covering the entire computer field.

Computer's popular departments include New Products, Product Reviews, Book Reviews, Standards, Open Channel (a reader forum), and Computer Society News.

Other membership benefits include the following:

- Discounts of up to 50% on **Computer Society Press books and videos.** Over 700 titles covering a broad spectrum of computer science topics including networking, communications, advanced systems, image processing, security, artificial intelligence, and visualization
- Discounts on registration to over 100 conferences, workshops, and tutorials each year
- Low subscription rates on special-interest periodicals
- Opportunity to participate in any of our 32 technical committees—networks of professionals with common interests in computer hardware, software, and applications
- Opportunity to participate in the development of more than 200 standards projects sponsored by the society
- Low member rates on COMPMAIL, the electronic mail service especially designed for computer professionals

Schedule of Fees

To join: see item 1, 2, or 3.
To subscribe: see item 4.

Membership dues and periodical subscriptions are annualized to, and expire on, December 31. Pay full- or half-year rate depending on date of receipt by the Computer Society as indicated below.

PRICES EXPIRE 12/31/92

	Half Year Mar 1-Aug 31	Full Year Sept 1-Feb 28
1 I don't belong to the IEEE and I want to join just the Computer Society	☐ $27	☐ $ 54
2 I want to join both the Computer Society and the IEEE*		
I reside in Region 1-6 (United States)	☐ $58.50	☐ $117
I reside in Region 7 (Canada)	☐ $53.50	☐ $107
I reside in Region 8 (Europe, Africa, or the Middle East)	☐ $53.00	☐ $106
I reside in Region 9 (Latin America)	☐ $49.50	☐ $ 99
I reside in Region 10 (Asia and Pacific)	☐ $48.50	☐ $ 97
3 I already belong to the IEEE and I want to join the Computer Society	☐ $11.00	☐ $22
IEEE Member Number _____		

*ACM members who join both IEEE and the Computer Society may deduct $5 off the full-year rate; $2.50 off the half-year rate.

4 SPECIAL-INTEREST PERIODICALS for new or current members

	issues per year		
IEEE Computer Graphics and Applications	6	☐ $12.50	☐ $ 25
IEEE Design and Test	4	☐ $11.00	☐ $ 22
IEEE Expert	6	☐ $10.00	☐ $ 20
IEEE Micro	6	☐ $11.50	☐ $ 23
IEEE Software	6	☐ $12.50	☐ $ 25
Transactions on:			
Computers	12	☐ $12.00	☐ $ 24
Knowledge and Data Engineering	6	☐ $ 9.50	☐ $ 19
Parallel and Distributed Systems	6	☐ $ 9.50	☐ $ 19
Pattern Analysis and Machine Intelligence	12	☐ $12.00	☐ $ 24
Software Engineering	12	☐ $11.00	☐ $ 22
IEEE Annals of the History of Computing	4	☐ $ 8.00	☐ $ 16

☐ Visa ☐ MasterCard ☐ American Express

Charge Card Number (Minimum Charge $10)

Mo. Yr.

Exp. Date

Checks drawn in local currency on a bank in the country of origin accepted from members in Australia, Austria, Belgium, Canada, Denmark, France, Germany, Ireland, Italy, Luxembourg, Netherlands, Norway, Portugal, Spain, Sweden, Switzerland, UK, USA, and Yugoslavia.

Periodicals total $_____

CA, DC residents add applicable sales tax $_____

Membership fees $_____

Total $_____

CANADIAN residents add 7% GST; BELGIAN residents add 6% VAT $_____

Total enclosed $_____

I hereby make application for Computer Society membership and, if elected, will be governed by IEEE's and the society's constitutions, bylaws, and statements of policies and procedures.

MAILING ADDRESS

Full signature _____ Date _____

☐ Male ☐ Female

First name _____ Middle initial(s) _____ Last name _____ Date of birth _____

Street address _____ City _____ State/Country _____ Zip _____

EDUCATION (highest level completed) Course _____ Degrees received _____ Date _____

Name of educational institution _____

OCCUPATION Title or Position _____

Firm name _____

Firm address _____

City _____ State/Country _____ Zip _____

ENDORSER (an IEEE member, Senior Member, or Fellow)

Endorser's signature _____

Name (print in full) _____ IEEE Member No. _____

Street address _____

City _____ State/Country _____ Zip _____

Mail or fax to appropriate Computer Society office:

EUROPE: 13, Avenue de l'Aquilon, B-1200, Brussels, BELGIUM. Fax: 32-2-770-8505
PACIFIC RIM: Ooshima Building, 2-19-1 Minami-Aoyama, Minato-ku, Tokyo 107, JAPAN. Fax: 81-3-3408-3553
ALL OTHERS: 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720–1264 USA. Fax: 714-821-4010

MICRO 4/92

Call for Papers

IEEE Micro seeks general interest manuscripts for 1992 and 1993 issues.

Suggested topics include multiprocessing, optical and biological computing, microcomputing to aid the handicapped, systems design, DSP-based tools, solid-state memory, and fuzzy-logic chips.

Submit six copies of papers to:

Editor-in-Chief Dante Del Corso
Dipartimento di Elettronica
Politecnico di Torino
C.so Duca degli Abruzzi, 24
10129 Torino, Italy
e-mail: delcorso@polito.it

or

Associate EIC Ashis Khan
Mips Computer Systems, Inc.
950 DeGuigne Dr.
Sunnyvale, CA 94086
phone (408) 524-7171
e-mail: ashis@mips.com

For author guidelines, contact Claire Azada IEEE Computer Society West Coast Office, phone (714) 821-8380 or fax (714) 821-4010.

Software Report

ISA in Taiwan

David K. Kahaner

US Office of Naval

Research, Far East

kahaner@cs.titech.ac.jp

I joined 140 scientists for the Second International Symposium on Algorithms at the Academia Sinica in Taipei, Taiwan, last December. The Academia is a government-funded institution that conducts scientific research and coordinates the efforts of other government research institutes and universities. Its Institute of Information Science organizes ISA with help from the National Tsing Hua University and the Special Interest Group on Algorithms of the Japan Information Processing Society.

ISA provides a forum for Pacific Rim researchers (and others) to exchange ideas on computing theory. Most of the attendees came from Taiwan, the US, and Japan.

Discrete algorithms. The participants were concerned almost exclusively with discrete algorithms. Papers focused on sorting, permuting, and the discrete aspects of computational geometry, combinatorial optimization, and graph traversal. Only a few papers dealt with parallel or distributed algorithms.

Potential applications abound, but ISA papers actually emphasized theoretical aspects. Coming from a mathematics background, I felt right at home with the tone of the papers, although I was unfamiliar with the techniques. Most papers presented elegant theorems analyzing discrete algorithms, data structures, or generalized graphs and expressed results as order or other limiting relations. There was very little evidence of actually using a computer except perhaps to experiment in new directions or verify an analysis.

To succeed in such research, all one needs is capable scientists; equipment and other expensive facilities are secondary. This is one reason that ill-equipped research institutes–not the case for this host institution–sometimes concentrate on these areas. Thus, it is not surprising that there is an almost seamless flow of research results in this field moving around the world.

Among my favorite papers were those on

- constructing the shortest watchman routes in a polygon, by Inagaki et al. of Nagoya, Japan;
- 3D channel routing for VLSI that tries to minimize the number of connections between different levels of the configuration, or *vias,* by Ho of Academia Sinica;
- path algorithms for robots minimizing total distance traversed, by Chan et al. of Hong Kong (one of a series of very excellent papers on geometry);
- a problem with testing logic circuits by applying a limited, selected set of test inputs, by Ibaraki et al. of Kyoto; and
- using a hypercube with faulty nodes to run an algorithm requiring a full binary tree, by Chan et al. of Hong Kong. I was impressed last year with related work from these authors.

Springer-Verlag offers the conference proceedings as "ISA 91 Algorithms," *Lecture Notes in Computer Science,* No. 557, W.L. Hsu and R.C.T. Lee, editors, 1991.

Academia Sinica. Our host, Taiwan Academia Sinica (Academy of Science), comprises 16 research institutes and preparatory offices for four more. The Academia was founded in 1928 on the mainland and moved to Taiwan in 1949.

Ta-you Wu, the Academia's president since 1983, was recently voted one of the two most popular men in the country. He earned his PhD at the University of Michigan and later chaired the Physics Department at the State University of New York at Buffalo.

0272-1732/92/0600-0069$03.00 © 1992 IEEE

Wu told me Taiwan divides responsibility for science and technology three ways. The Ministry of Defense researches military technologies. The Ministry of Economic Affairs oversees technology related to industrial development. The National Science Council supervises academic programs for basic, applied, and social science. The Academia, however, isn't under any of these three organizations and reports directly to Taiwan's president. *[Software Report, June 1991, elaborates on Taiwan's computer industry. – ed.]*

Institute of Information Science. IIS's computing facilities include an Ncube, two Iris graphics workstations, about 50 other Unix workstations, and plenty of PCs. Researchers have access to an ETA-10. A computer vision lab offers image scanners, image processors to support 2D animation and 3D visualization work, and facilities to support stereo vision and neural network research.

The Ncube offers opportunities for parallel processing research related to architecture design, compilers, and parallelizing various constructions within existing languages. Robotics research at IIS focuses on dextrous manipulation, coordinated motion of multiple robot arms, and simulation.

About a half-dozen researchers work on the theoretical aspects of discrete (combinatoric) algorithm development. There are significant efforts in real-time operating systems and high-speed networking and software methodologies, and a small effort on VLSI layout design.

TRON show in Tokyo

At last November's TRON show in Tokyo about 20 vendors displayed products and applications using TRON standards, which feature a coherent design for applications ranging from embedded systems to large-scale distributed computer systems. Most of the big Japanese computer manufacturers exhibited TRON products, except NEC, which successfully markets MS-DOS/Windows/Unix machines and has developed its own series of Intel-compatible microprocessors, the V-series.

Several full lines of 32-bit microprocessors supporting TRON standards are available, and the next-stage 64-bit processors are expected. Interest is rising in the use of ITRON and myITRON in multifunctional fax, video recorders, video cameras, and printers, using MPUs with TRON specifications and non-TRON processors. Business applications, where BTRON competes with existing Unix software in business applications, are still in experimental stages. Multimedia applications, however, should help promote BTRON's real-time potential.

To succeed in theoretical research, all one needs is capable scientists; equipment and other expensive facilities are secondary.

In addition, the TRON standards may receive a boost from Ken Sakamura's concepts of a TRON house, building, and computer city in Chiba Prefecture. Sakamura, an associate professor of information science at Tokyo University, originated the TRON concept.

Software. At the show, Fujitsu demonstrated its real-time operating system, based on ITRON specifications, called REALOS/Gmicro. Oki showed its real-time operating system RG68KS, based on CTRON, TRON's communications specification.

A multimedia working group has proposed a system connecting BTRON workstations via an Integrated Services Digital Network (ISDN) to form a conference system. US-based Wind River Systems brought its real-time operating system VxWorks on Gmicro processors.

Microprocessors. Fujitsu presented the Gmicro G32 series featuring 32-bit MPUs. The top-of-the-line 300-version achieves 24 MIPS at 33 MHz. The company also plans 400- and 500-versions and offers a range of peripheral chips. Hitachi presented a similar program, its H32-series, and Toshiba is developing a line with a 32-bit TX1 processor.

Genesys has developed a multimedia board based on an F32/300 (32 MIPS) processor, that uses the TRON Application Database (TAD) to handle all kinds of media. And Matsushita showed its MN10400 32-bit MPU that runs 20 MIPS for fast execution of TRON commands.

Other applications. The electric machinery maker Meiden demonstrated a workstation for factory automation. The machine, based on NEC's 32-bit MPU V80, runs NEC's RX-UX8322 real-time Unix, which supports Unix V and ITRON as its kernel. It provides peripheral boards for multichannel communication and Meiden Real-Time Basic (MRTB) for parallel execution. Nihon Minicomputer produces a simpler system called TB-100 that uses a Gmicro/100 processor and runs ITRON.

Mitsubishi Electric showed a fax machine running myITRON and a color copier with an outline font driver using Gmicro processors. Personal Media demonstrated a notebook computer (based on Matsushita's implementation of the Intel 386) with BTRON as the operating system and a window interface. The company also developed a workstation using TRON microprocessors and an operating system based on BTRON specifications, B2. Japan Airlines accesses its passenger reservation system through BTRON terminals, and Matsushita introduced its Educational Computer based on BTRON1 specifications, called PanaCAL ET.

[For more information on TRON, see the IEEE Micro *special issues on TRON 1987-1991 and "The TRON Intelligent House,"* IEEE Micro, *April, 1990.– ed.]*

Human interface project

Friend 21 is a project of Japan's Ministry of International Trade and Industry (MITI) to develop human-computer interface technology. The national project is administered from a central institution within MITI and sponsored by 14 companies, including computer manufacturers, home electronics corporations, and publishing houses. The International Symposium on Next-Generation Human Interface in Tokyo last November drew 600 participants (mostly Japanese) interested in the project and presenters from Japan, the US, Canada, and Europe.

The personal environment (PIE) that is the goal of the effort targets the untrained casual user rather than the professional. A model presented by Hirotada Ueda of PIE/Hitachi and Hajime Nonogaki of Fujitsu brought together users in a "studio environment," from which they could access "newspapers," "video," and a "database."

An open shared workspace designed by Hiroshi Ishii of NTT Human Interface Laboratories intended to overcome acceptance problems by not forcing users in a completely new environment. Yuzuru Tanaka, a professor at Hokkaido University, showed an impressive video of his Intelligent Pad system, which relies on a generic tool kit, synthetic programming, open platform, and integrated management. In it, objects could easily be combined, cut into pieces, and rearranged.

Two companies presented results in accessing multimedia. Miyatake of PIE/Hitachi showed an advanced digitized video tape editor that includes automatic shot separation, iconization, and editing tools. Watanabe of PIE/Sony explained automatic shot separation and investigations of TV quiz programs for development of scenario-based interfaces.

The final panel discussion on "Human Interface in the Future" was led by Mario Tokoro, a professor at Keio University who is also affiliated with Sony. In his introduction he gave a picture of a "sea of computers" where people can move freely, contacting others with a pocket-size computer. Among the other panelists, two approaches were apparent. One was the idea of making computers useful for everyone in society. The other focused on identifying opportunities for computers to support or replace human activities. A philosopher from Chiba University, Shun Tsuchiya, concluded that, rather than focus on interfaces, we should educate our children about their responsibilities when using computers. Children must learn, Tsuchiya said, that computers may be faulty or tempt us to infringe on others' personal data.

MITI's Real World Computing program explores flexible information processing.

Soft logic program

MITI's Real World Computing (10-year advanced computing) program aims not to develop a new computer but to explore basic technologies thought to be significant to the general area of flexible information processing. Flexible information processing, or soft logic, is the logic system carried out (unconsciously) by humans.

Several forms of new computer technology, including optical, neural, or massively parallel, may provide the computational basis for the actual work to be done. System integration is a key aspect of the work. The computational hardware will provide the tools for higher level theoretical foundation work related to the basic theory of flexible information processing. This in turn will allow research in higher level (but still elemental) functions such as recognition, understanding, inference, problem solving, autonomous and cooperative control, simulation, and human interface. MITI hopes these functions can be integrated in an advanced way to provide genuinely flexible information processing.

MITI admits that it does not know the right approach to many of the problems it wants to resolve. Therefore, in the first half of the program, competitive research teams will work towards the same targets, their methods and successes to be evaluated after five years. The structure of the research organization establishes a central laboratory and several distributed laboratories. The center will research common themes and integrate the results of research by the others.

Foreign firms and individuals can participate in the program by paying an initiation fee and joining the RWC Partnership. The partnership, which will be partially funded by MITI, will generate research plans, provide the R&D infrastructure, manage subcontractors, and carry out research.

[David Kahaner is on assignment with the US Office of Naval Research, Far East. His comments are his own; they do not express any official policy.]

Meet the experts

Richard Mateosian

2919 Forest Avenue

Berkeley, CA

94705-1310

(510) 540-7745

Tog on Interface, Bruce "Tog" Tognazzini (Addison Wesley, 1992, 347 pp., $26.95)

Bruce Tognazzini's official job title at Apple Computer, Inc. is Human Interface Evangelist. Apple makes its money selling metaphors and has given Tognazzini this title as a metaphor. An evangelist is one who carries the gospel (good news) of the Christian religion to all who will listen. Tognazzini's job is to carry the good news of the recent advances in human-computer interface design to anyone who will listen.

Tognazzini is one of Apple's earliest employees. He worked on the first version of the Apple Human Interface Guidelines more than 10 years ago. Since March 1989, he has written a monthly question-and-answer column in *Apple Direct*, a publication that Apple targets at software developers. He gathered material from these columns and reworked it into this book.

I liked this book. Tognazzini has lots of experience and insight. He preaches extremely important principles. Nonetheless, he has written many heavy-handed passages. He takes his evangelical responsibilities seriously. He targets his message at everyone, not just those who are ready, willing, and able to hear it. Sometimes this leads him to try to shout down opponents whose positions are not as untenable as he makes them out to be.

A prime example of his overexuberance occurs in the discussion of mousing vs. keyboarding for making menu selections. He cites studies that prove that mousing is faster. He acknowledges that almost everyone thinks keyboarding is faster, but he attributes this to a perceptual illusion. In exchanges of letters on the subject he ridicules his correspondents' positions. Fortunately, his style is good natured and free of malice, so no one is likely to take offense.

I am especially fond of one of the themes of Tognazzini's book. He points out that software developers are not usually competent graphic designers or writers. He suggests hiring professionals to perform these functions. As a professional writer, I have often experienced a phenomenon that he describes. In striving to understand a product well enough to write about it, I uncover problem areas. Similarly, a graphic designer can often identify inconsistent or clumsy aspects of a product's visual interface. Often, the developer's best course is to treat these areas as bugs to be corrected rather than features to be explained.

Tognazzini also emphasizes the importance of testing user interfaces by watching actual people trying to learn to use them. His best stories about user testing are funny, because they show how different a user's reaction can be from what the developer expected. For example, he worked on a program called *Apple Presents...Apple, an Introduction to the Apple II Plus Computer.* The program needed to determine whether the attached video monitor was color or monochrome. It did so by asking the user, while it displayed a color graphic on the screen. Users with color monitors were able to report the fact correctly. The designers went through five unsuccessful versions before they hit upon "Do the words above appear in several different colors?," which users with monochrome displays would always answer correctly.

Tognazzini rushes fearlessly into some areas that more cautious writers might consider too speculative. One interesting example is in his chapter, "Carl Jung and the Macintosh." Jung divided people into two kinds on each of four axes, resulting in a typology that places each person into one of 16 boxes. Isabel Myers-Briggs

0272-1732/92/0600-0072$03.00 © 1992 IEEE

developed and popularized this idea, so that nowadays one's Jungian type is as good a conversation starter in some circles as one's zodiacal sign. Tognazzini's type, for those who know how to interpret these things, is INFP, while mine is INTJ.

According to Tognazzini, only two of the four Jungian axes have any relevance for human-computer interfaces. These are the introvert-extrovert (I-E) axis and the intuitive-sensory (N-S) axis. According to Tognazzini's study of Apple employees, most engineering personnel fall on the N side of the N-S axis, while the vast majority of the population fall on the S side. Similarly, about twice as many of the engineering personnel fall on the I side of the I-E axis as in the general population. Oversimplifying and extrapolating this result, a small cadre of INs are designing interfaces to be used by the great multitude of ESs. The INs are separated from external reality, depend on their own internal model of reality, can shift rapidly among levels of abstraction and tie together past and immediate experiences. The ESs live in the reality of their immediate sensations, prefer the concrete to the abstract, and don't tie together experiences that occur at different times. What a mismatch!

Tognazzini emphasizes again and again the importance of making the artificial reality of the human-computer interface consistent and believable. Of course, he knows which side his bread is buttered on, so he soft pedals his criticisms of Hypercard, an Apple product that blatantly ignores many of the principles that he espouses. For example, he feels developers should not hide the menu bar without a powerful, overriding reason, and that they should make all icons require double clicking. As anyone familiar with Hypercard knows, its developer did not follow these rules.

I liked Tognazzini's description of a video he made in which he contrasted the engineering-oriented basement of a hotel with the user-centered world of its lobby. He sits at a table in the lobby, absentmindedly pouring salt into his coffee as he pontificates about interface design. Suddenly a dialog box appears in the air informing him that pouring salt into coffee was an unexpected event that has caused the lobby to "quit." He is back in the basement, holding coffee cup and salt shaker and looking bewildered.

Tognazzini emphasizes that artificial reality of the human-computer interface must be consistent and believable.

This example catches the essence of the problem with the Macintosh interface. Unlike users of Unix or MS-DOS, Macintosh users are completely unfamiliar with the basement. When the lobby quits, they have no recourse. The lights in the basement are off, and they don't even have a flashlight. Developers of Macintosh applications must be extremely careful to preserve the artificial reality they've created. I can't advise developers on how other users will react, but I have a hard and fast rule. When I try a piece of software and it gives me one of those "bomb" dialog boxes, I immediately take it off my machine, and I never use it again.

Tognazzini touches on many other interesting subjects. I can't go into his discussions of object-oriented programming or his speculations on why he found himself bumping into walls and furniture after returning from a camping trip. I won't tell you his opinion of the convention that allows Macintosh users to eject diskettes by dragging their icons to the trash can icon. For these subjects, you'll have to read the book yourself. I enjoyed it, and I think that you will too.

Dan Gookin's PC Hotline, Dan Gookin (Microsoft Press, 1992, 252 pp., $14.95)

Microsoft has done a good job backing up their software products with books. Users of Word, Excel, Windows, and DOS can choose among many well-written books. Some are for naive users, some are for experts, and some are for developers. Most are comprehensive, accurate, carefully edited, and nicely published. I'm sure Microsoft realizes that selling someone a book is a lot more profitable and effective than answering their phone calls.

Gookin's book makes that trade-off explicit. Microsoft urges you to buy the book rather than pick up the phone when problems arise. This is more than a little optimistic on their part. On the other hand, users who have read Gookin's book before running into trouble will probably be able to turn to the information they need when the time comes.

Gookin addresses performance problems, crashes, and viruses. He emphasizes preventive medicine and emergency preparedness. Problems are less likely to occur, and you will be better able to deal with the ones that do, if you follow his recommendations. The time to make an emergency boot disk is before your hard disk fails to boot. The time to document the contents of your battery-powered CMOS memory is before you have to restore it.

Gookin's book is full of nuts-and-bolts facts and advice. Most of it is pretty dull, but you'll be glad to have it when you need it. If you're the PC expert at your installation, get this book and read it before the big one hits.

American Men and Women of Science, 18th ed., 1992-93 (Bowker, 1992, 8 volumes, 8,498 pp., $750)

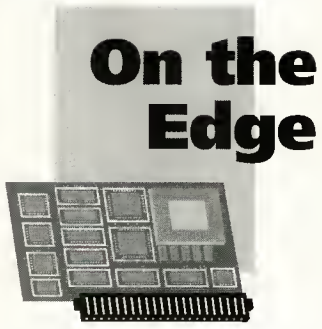
In my August 1989 column I reviewed the 17th edition of this work, and I have approximately the same reaction this time. If you need the kind of information it contains about one of the 122,000 scientists who happen to be listed, this is as good a way to find it as any I can think of. My complaint now, as then, is that its coverage of computer and information science is scanty and that their selection criteria seem haphazard.

For example, the compilers of this work have included neither of the authors whose works I reviewed in this column. I looked up the first 122 names (all of the As and Bs) from the April 1992 *Directory of Volunteer Leaders and Staff of the IEEE Computer Society.* I was able to find only 29 of them in this work. I'll let you come to your own conclusions about the significance of those statistics.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 183 Medium 184 High 185



Carl Warren

McDonnell Douglas

Space Systems Company

(714) 896-3311

x. 7-1230

warren@ssdunx.mdc.com

Object encyclopedia technology

[I invite readers to send me information on a tool or method that solves problems for consideration in future columns.—C.W.]

Lee Neitzel, CTA Incorporated

The shift to heterogeneous distributed processing environments has intensified the problem of accessing remotely located data and services. This problem exists in both automated offices and factories. For example, in automated factories not only are information systems distributed, but so are control systems. And, to make the problem even more complex, factory information systems are being integrated with their control systems.

The US National Aeronautics and Space Administration (NASA) and McDonnell Douglas Space Systems Company initiated a solution to this problem. CTA Incorporated participated with the Instrument Society of America (ISA) and the International Electrotechnic Commission (IEC) to bring about a standardization of the solution. Our approach defines and standardizes an extensible set of object definitions known collectively as an encyclopedia.

The object definitions in an encyclopedia provide information about both instances of objects and collections of these instances. They contain information that can be either generic to a class or specific to an instance.

We defined the encyclopedia standards[1] to specify in detail each piece of information used to define objects, such as names, attributes, and methods. Each piece of information represents a particular aspect of an encyclopedia object model, which is also specified by the standards.

Encyclopedia objectives

The encyclopedia object model forms the basis for organizing and structuring object definition information. We designed it to meet four primary objectives.

First, we wanted to define related, but different, types of objects, such as devices, device managers, application processes, and communication protocols. To provide this capability, we adopted an object-oriented model with inheritance.

Second, we wanted to specify the components of an object. For example, we can define a simple device to contain a device manager and one or more application processes while defining a

0272-1732/92/0600-0074$03.00 © 1992 IEEE

more complex device to include subobjects that are themselves devices. To provide this capability, the encyclopedia standards support an entity relationship model that allows us to define relationships between objects, such as containment.

Third, an encyclopedia must contain information about how to access an object. Further, this information, or a subset of it, should be downloadable to a directory. More precisely, the encyclopedia standards should let us specify an object's functional interfaces (methods), the communication protocols used to access them, and additional access information, such as the object's network address.

To provide these capabilities, the encyclopedia standards specify how to define object interfaces and how to relate them, parameter by parameter, to the services of communication protocols. The provision of this capability includes the abstract syntax definition of data structures passed between objects.

To provide for integration with directory systems, we defined the encyclopedia standards to be consistent with the CCITT/ISO 9594 X.500 Directory Standards. For example, we can tag specific object attributes, such as a network address, as directory attributes and export them to the directory system.

Fourth, we wanted to exchange object definitions across a network. Therefore, we based the encyclopedia standard on schema as opposed to language, and specified them using ASN/ISO.1. The secondary benefits of this approach are that language-based schemes such as the IEEE P1175/D11[2] standard can provide user interfaces for the specification of objects, and the encyclopedia can be used as the language-independent information repository.

Encyclopedia technology

McDonnell Douglas and CTA developed an encyclopedia tool based on the encyclopedia standards. We use this tool to integrate definitions of the objects being developed by McDonnell Douglas for the NASA Space Station.

This tool lets us standardize definitions of objects and functional interfaces, and classify and associate data to create a knowledge base about Space Station objects.

The encyclopedia knowledge base and its associated software tools can now produce reports, analyze configurations, and generate code. Reports, such as bills of material, functional decomposition, and data and control flows, integrate and disseminate design information from multiple sources.

We use automatic configuration analysis in the integration process for such tasks as verifying interfaces and evaluating end-to-end data flows.

Code-generation capabilities allow export and import routines to be automatically generated from interface specifications and linked to application code that has been generated from object behavior specifications.

These capabilities will permit the generation of code to

1) exchange data between databases, commercial tools, and commercial systems such as X.500 Directory systems; and
2) test application and protocol interfaces and operations.

Summary

Though heterogeneous networks have become commonplace, they have also complicated the development of systems expected to operate over them. Standards are under development for the definition of such systems. These standards support the construction of knowledge bases and automated tools.

Because the encyclopedia standards treat communication protocols and other commercial products with an application interface as objects, users can generate code for their application interfaces and link them with the generated application code.

The McDonnell Douglas Space Systems Company and CTA Incorporated have developed a set of automated tools that conform to these standards. These tools support both systems and software engineering disciplines from the design phase through system implementation, test, and operation.

References

1. *Field Bus Application Layer Specification–Object Model,* IEC 65CWG6-90-01-003-WD.
2. *Draft IEEE Trial-Use Standard Reference Model for Computing System Tool Interconnections,* Sep. 20, 1991.

Lee Neitzel is a computer systems engineer with CTA Incorporated and currently works on the Space Station Program. He holds an MS degree in computer science from George Washington University and has been active in a number of communication protocol standards committees.

A plan for the future

[I've invited Steve Diamond, MSC technical committee chair, to write about the TC's work, its importance to the Society in general, and the main issues it expects to face in the next few years.—C.W.]

Stephen L. Diamond, Chair, IEEE Microprocessor Standards Committee

Carl Warren

McDonnell Douglas

Space Systems Company

(714) 896-3311

x. 7-1230

warren@ssdunx.mdc.com

It has been said that one of the hallmarks of a mature organization is the need to plan for the future. Based upon this criteria, I believe the Microprocessor Standards Committee (MSC, a standards-creating activity of the IEEE Computer Society's Technical Committee on Microprocessors and Microcomputers) has become a mature organization. This year, the MSC inaugurated a strategic planning committee to help it understand what the future holds for us and how we will respond to these challenges.

The challenges that the MSC faces are numerous and nontrivial. Many of these challenges resulted from the success of the workstation and desktop market. This success has thrust the standardization of (or lack thereof) microprocessors and microcomputers—and their associated ancillary products—into sharp relief. We see two good results: The MSC is having an impact on the market as a whole, but the market has an impact on us also. I would like to examine both of these scenarios in slightly greater detail, and then tell you why the planning function has become so important to the MSC. Using the following story of a standard as a vehicle for this examination should help.

An MSC working group created a standard known as ANSI/IEEE Standard 754 for binary floating-point arithmetic. Subsequently another MSC working group followed with ANSI/IEEE Standard 854 for radix-independent floating-point arithmetic. Great amounts of work went into both of these standards—work that was both theoretical and experimental. This energy proved that the concept was sound and practical and demonstrated that these standards could be applied. Both standards resulted from substantial committee work, and both pushed the leading edge of technology and knowledge. While in draft form, both standards met opposition in committee from a few large vendors. However, over time, both were accepted by the IEEE and ANSI (American National Standards Institute) as standards for dealing with floating-point calculations.

And then the challenges to the standards began. Some of the major vendors of information technology (IT) products, specifically, the mini- and mainframe manufacturers, did not accept the standards. While the MSC had demonstrated that the standards could be applied to the processors in question, the standards were revolutionary in their approach to the question at hand. The vendors complained of limited backward-compatibility with an installed base that was approaching nearly a trillion dollars in value. Unfortunately, we did not convince these major IT vendors that they could implement the standards without imperiling their base.

As a result, the mainstream vendors never accepted the two IEEE standards. Instead, the ISO/IEC Joint Technical Committee 1, Subcommittee 22, Working Group 11, initiated a controversial international standardization effort. The US liaison to this committee was X3T2. (The Accredited Organization IEEE and X3 are peer groups in matters of standardization; X3 and the IEEE operate under ANSI rules and are subject to the same responsibilities.) After several years

0272-1732/92/0600-0076$03.00 © 1992 IEEE

of work, this committee (SC22 WG11) produced a Committee Draft that incorporates ANSI/IEEE Standards 754 and 854 but also incorporates the methods currently used by many of the major companies in the IT business.

This is one side of the coin—the need for evolutionary change to protect the installed base that grows every year. At the same time, the work of the MSC has become important to and embraced by many of the firms in Silicon Valley, firms that are dedicated to changing the way that the world perceives computing. The PC and the workstation received their starts here, and these two advances substantially changed the way that the world does computing. Most of these exciting new areas can look toward the MSC for their standardization needs.

While the MSC continues to sponsor PARs (Project Authorization Requests), we are finding that standards developing organizations (SDOs) other than the IEEE are becoming involved in the areas we've traditionally considered to be MSC-exclusive areas. We find that other SDOs are increasingly in conflict with the MSC over the new PAR areas, which tells me that what used to be solely MSC concerns are now industrywide concerns. This means that the MSC was right: These are areas of concern to the whole industry.

It is also the other side of the coin. We could be victims of the success of the industry we have helped to grow. Even more interesting, the nature of Silicon Valley is changing, with software becoming more and more important to the nature of the business.

This leads me to the final point that I wish to make. We know that standards are a change agent in the industry—as evinced by the market the IEEE 802 family of standards created for LANs. OSI has changed the market, as did FDDI (X3T9.5), WANs (IEEE 802.6), Posix (IEEE 1003), and SQL (X3H2). All of these standards helped change the way the industry does computing. Users demanded some of these standards for interoperability, and vendors pushed for others to permit better processing. All of them caused change, and all were ultimately accepted by the industry, which is composed of both users and providers.

I believe that understanding the needs of the user is the key to the survival of the MSC, and, in a larger sense, the entire standards discipline. The highly evolutionary (and occasionally revolutionary) nature of the IT industry poses the biggest opportunity and the biggest challenge for us, both as a society and as an industry. The users of MSC products 10 or even five years ago may no longer be our users today. We need to find out who our new users are and why they need MSC-produced standards. We also need to find out where the new challenges and opportunities will come from and find new places that need the skill set that the MSC can bring to bear. We need to learn how to produce standards that somehow can be translated into something that makes computing better, faster, or cheaper for these users. Our goal has been—and must continue to be—to increase the utility of the compute function for our customers.

This is why I have revitalized the strategic planning function of the MSC under Phil Huelson (who also chairs P1754, a RISC microprocessor architecture standardization effort). We need to know where we are going, and we need to know if the users are moving there with us. Several quotes come to mind in this situation, with the most famous being from *Alice in Wonderland*. When Alice asks directions of the Cheshire Cat, she admits that she doesn't know where she is going. The cat then tells her that any direction is right, for if you don't know what your destination is, any path suffices.

I would like to focus the MSC on the future over the next two years. My predecessor, Clyde Camp, left me a recognized organization that is ready and willing to move into different areas. We are aggressively pursuing new standardization options. For example, the MSC sponsored the newly approved IEEE Std. 1596-1992 Scalable Coherent Interface (SCI) under David Gustavson. However, to survive, we must continue to grow and change. Our industry has changed; our challenge during the next several years is to understand the change and to make the change work for the MSC, for our users, and for the industry as a whole. I ask that all interested parties join us in this important revitalization effort.

For information about participating in the IEEE MSC, contact Steve Diamond, c/o Sherrie Bolin, SunSoft, Inc., 2550 Garcia Avenue, M/S MTV08-221, Mountain View, CA 94043; phone (415) 336-4190, fax (415) 336-4477, or e-mail steve.diamond@eng.sun.com; Compmail: s.diamond. SunSoft is a Sun Microsystems operating company.

For information about participating in the IEEE MSC Strategic Planning Committee, contact Phil Huelson, c/o Sparc International, 535 Middlefield Road, Suite 210, Menlo Park, CA 94025; phone (415) 321-8692, fax (415) 321-8015, or e-mail phil@sparc.com.

Stephen L. Diamond chairs the IEEE Computer Society Microprocessor and Microcomputer Standards Subcommittee, of which the MSC is the executive committee. He also serves as director of standards at SunSoft, Inc., where he is responsible for the company's participation in worldwide standards activities.

IEEE MICRO

Information for Authors

February 1992

Who we are

IEEE Micro, a bimonthly publication of the IEEE Computer Society, reaches an international audience of microcomputer and microprocessor designers, system integrators, and users. Readers seek to increase their technical knowledge of computers and peripherals; systems, components, and subassemblies; communications, instrumentation, and control equipment; and software.

What we publish

IEEE Micro publishes original works about 5,500 words long (about 20 double-spaced typed pages that include explanatory figures, tables, and programs). These works discuss the design, performance, or application of microcomputer and microprocessor systems. Readers welcome tutorial material, review papers, and discussions of standards. Topic areas include:

- systems
- fault tolerance
- languages
- application software
- algorithms
- architecture
- data acquisition
- operating systems
- artificial intelligence
- communications
- hardware and software design and implementation

Submitting your manuscript

Submit six copies of your manuscript and a 50-word abstract with keywords along with your mailing address, phone and fax numbers, and electronic mail address directly to:

Prof. Dante Del Corso
Editor-in-Chief, *IEEE Micro*
Dipartimento di Elettronica
Politecnico di Torino
C.so Duca degli Abruzzi, 24
10129 Torino, Italy
Telephone: + 39 11 564 4044; fax: + 39 11 564 4099
Compmail: d.delcorso; Bitnet: delcorso@itopoli;
Internet: delcorso@polito.it
or
Ashis Khan
Associate Editor-in-Chief, *IEEE Micro*
Mips Computer Systems, Inc.
950 DeGuigne Drive
Sunnyvale, CA 94086
(408) 524-7171
Internet: ashis@mips.com

All manuscripts pass through a peer-review process consistent with other professional-level technical publications. This process may take up to four months, and referees may require revisions to parts of your work. If a manuscript exceeds the specified length, it will be shortened.

Successful contributions avoid the style of transactions and academic journals. They sufficiently introduce the material, place it in context with similar works, describe the practical or potential applications of the material presented, and discuss both pros and cons of the approach. At least 20 percent of the article is tutorial in nature. Brief literature surveys do not satisfy this requirement.

After accepting your manuscript for publication, the editor-in-chief will ask you to supply three copies of any revised draft, plus drawings, photographs, equations, and programs; an electronic version; and biographies and photos of all authors. In addition, you must sign a release transferring copyright to the IEEE (excepting certain key rights retained by the author).

Submit the hard copies, including illustrations and references or bibliographies, printed on one side only of 8 1/2 × 11-inch paper and double spaced with at least 1 1/2-inch margins. Send an electronic copy on floppy disk or via Compmail or Internet. All electronic files should retain any text-formatting codes you use and identify the formatter used. Refer to the Computer Society's Electronic Submittal Guide

for further details. Disks must be Macintosh-compatible or 5.25-inch, IBM PC-compatible, and running DOS Version 2.10 or newer. For further guidance, contact:

Marie English
Managing Editor, *IEEE Micro*
10662 Los Vaqueros Circle
PO Box 3014
Los Alamitos, CA 90720-1264
Telephone: (714) 821-8380; fax: (714) 821-4010
Compmail: m.e.english

Professional editors on the *IEEE Micro* staff thoroughly edit accepted manuscripts. This collaborative process between author and editor results in a concise, well-worded article.

Writing tips

Readers welcome clear, accurate articles presented in logical sequence. Let readers know in the first paragraph why your subject is important; give them a reason to continue reading. Augment your discussion with examples, tables, diagrams, charts, and photographs to help readers grasp your point. Remember, all readers won't be familiar with your specialty; you will have to explain unusual terms or intricate processes.

Readers move swiftly through articles written in the active voice and containing short words, short sentences, and concrete examples. (An active voice example: "This scheme contains two main buses" NOT "Two main buses are contained in this scheme.") Avoid jargon, explain acronyms, and simplify your language. For example, use "to" NOT "for the purpose of" and use "can" NOT "has the capability to." In other words, write the way you talk.

As you can see, magazine style differs from journal and report styles.

References and bibliographies

References substantiate points made in the text or cite previous or important works. Do not overdo it, however; most articles need less than 15 citations. They appear in numerical order in the article and in a separate section at the end of the article. Citations in the text appear as Arabic superscripts, for example, Smith.[1]

Cited sources should be available to the reader; don't include unpublished works. Any abbreviations should follow *IEEE Micro* usage; see a recent issue for examples. When in doubt, spell it out.

You should attempt to provide full bibliographic data as a courtesy to your readers. A complete citation includes author(s); title of article or chapter; title of journal, book, proceedings, or dissertation; volume; number; publisher's name, city, and state for books and dissertations; complete address for private technical reports; year published; and inclusive page numbers.

Illustrations

Submit photocopies of illustrations, rather than originals, for the initial manuscript review. Cite permission for any previously published images or figures so that *IEEE Micro* can properly credit the source. (See Figure 1.)

All illustrations and drawings should be clear and submitted in hard copy (on separate sheets) and on Macintosh-compatible electronic disks where possible. Photographic prints should have good contrast and gradation and should be at least 3 × 5 inches in size. Number, caption, and cite in text all illustrations and tables. Check to see that all artwork is accurate and unambiguous, and uses the same terms as the text.

IEEE Micro reproduces your original halftones, machine-made graphs, computer printouts, and electronically produced artwork. Artists will redraw all other art to meet house standards.

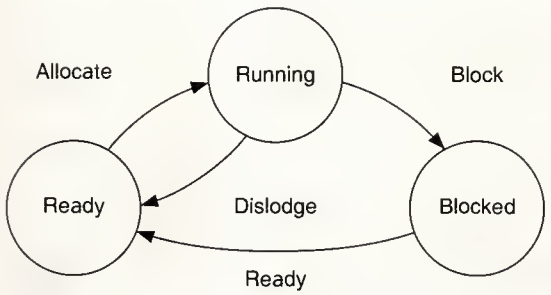


Figure 1. Task states and transitions. *(Copyright 1995 William Jones. Reprinted by permission.)*

Biographical sketch and photograph

Submit a photograph and biographical sketch of each author. Good-quality, black-and-white glossy photographs, preferably 3 × 5 inches in size, reproduce best. Limit biographical sketches to 75 words and include, in the following order: current positions and technical interests, prior professional experience and other important activities, education, professional affiliations, and current address. See a recent issue of *IEEE Micro* for examples.

Micro News

continued from p. 9

IEEE Micro volunteers honored

Carl D. Warren, Associate Editor of *IEEE Micro*, has been awarded the Meritorious Service Certificate. The IEEE Computer Society Awards Committee approved the award on May 11, to recognize his outstanding service to the magazine since 1989. The award recognizes a volunteer's significant contribution to a Computer Society activity, based on excellence, dedication, and tenure. Warren supplied material for the On The Edge and Micro Standards columns and actively sought the participation of other contributing authors.

Intel has named former *IEEE Micro* Editorial Board member **John Crawford** as an Intel Fellow, the company's highest technical position. As chief architect for the Intel 386 microprocessor, Crawford defined the company's 32-bit architectural extensions to the 8086/186/286 16-bit product line. He also contributed to the 486 family. Currently he comanages development of the company's next-generation microprocessor product, scheduled for release later this year. This processor contains more than 3 million transistors and computes 100 MIPS.

Intel has appointed only seven fellows in its 24-year history, six of whom are still with the company. Fellows are encouraged to explore new directions in technology and choose the research they will conduct.

NIST offers R&D grants

The National Institute of Standards, under the US Department of Commerce, awarded more than $90 million in its Advanced Technology Program, intended to assist businesses with research and development on precompetitive, generic technologies. The 27 grants support research and development to resolve technical uncertainties and permit an assessment of commercial potential, prior to development of commercial applications.

NIST began the program in 1990 to fill a perceived gap in technology commercialization. According to NIST director John Lyons, basic research and product development can find funds. But the in-between stage, in which researchers bring technology to a point where industry can begin to develop specific products, needs support.

Grants are limited to commercial ventures; universities and government agencies may receive funding only by participating in an industry-led joint venture. Awards to individual firms are limited to $2 million over three years and can be used only for R&D costs. Joint ventures may be funded for five years.

Among this year's grants was a $928,000 award to the National Storage Industry Consortium to develop a data recording head capable of 10 Gbits per square inch–100 times more than today's best commercial devices. Iterated Systems received $663,000 to develop a digital image storage and decompression chip using fractal transform image compression technology. A consortium comprising Honeywell, Hercules Aerospace, Sheldahl, and 3M received a $660,000 grant to develop sensor and control technology based on neural networks for application to complex materials processing.

The department solicits proposals once a year. Proposals are evaluated on their potential for broad-based benefits, technology transferability, the proposer's qualifications, commitment, and organizational structure. Semifinalists make oral presentations, and evaluators may visit a site to assess facilities. Foreign firms are restricted, but may qualify based on the discretion of the Secretary of Commerce.

For more details contact the Advanced Technology Program, A430 Administration Building, NIST, Gaithersburg, MD 20899; or call (301) 975-2636. An ATP hotline at (301) 975-2273 offers a status report on the program.

University aims to graduate more women engineers

The University of Texas at Austin has established a Women in Engineering program to boost women's enrollment

Micro Bits

The public review and comment period on X3.222-199x, high-performance parallel interface physical switch control (**HIPPI-SC**) extends through July 6, 1992. The standard can be purchased from Global Engineering Documents, PO Box 19539, Irvine, CA 92713-9539, for $25 (domestic), $32.50 (international). Send comments to X3 Secretariat, Attn: Lynn Barra, 311 First Street, NW, Suite 500, Washington, DC 20001-2178. Send a copy to American National Standards Institute, Attn: BSR Center, 11 W. 42nd Street, 13th Floor, New York, NY 10036.

VLSI Research, Inc. reports that the **semiconductor equipment market** fell 4 percent in 1991 to $8.1 billion. Wafer processing took the biggest hit, declining by 9.1 percent. However, the assembly market, due to a contract with the former USSR, was able to show a 17 percent increase over 1990.

A **shareware** version of Plotdata is available as PlotD11A.ZIP on directory SIMTEL20 at anonymous file-transfer protocol sites. With the program, users can view, edit, analyze, differentiate, integrate, calculate, and graphically reproduce data. An upgrade of Symbmath (Micro bits, Feb. 1992) that includes an expert system is also available as SM20A.ZIP on the same directory.

and graduation numbers in engineering. The program comes after a report by the campus' Women in Engineering Caucus stating that, although women make up more than half of the university population, they represent only about 15 percent of engineering BS recipients.

The new program will organize retention efforts including freshman seminars for women, increased participation of women in undergraduate research, and expansion of the mentor program. The program will also focus on recruiting efforts, including fund-raising for additional women's scholarships, outreaching to middle and high schools, and involving alumni in recruitment.

Ed board member returns

Victor K.L. Huang, manager of research and development of microelectronics systems and applications at the National University of Singapore's Institute of Microelectronics, has returned to the editorial board of *IEEE Micro*. He will review manuscripts for the magazine.

Huang earned a BS at the Virginia Military Institute and MS and PhD degrees at the University of Virginia, all in electrical engineering. He is a senior member of the IEEE, a member of both the IEEE Computer Society and Association for Computing Machinery, and a senior administrative committee member of the IEEE Industrial Electronics Society.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 198 Medium 199 High 200

New Products

Send announcements of new microcomputer and microprocessor products to Managing Editor, IEEE Micro, *PO Box 3014, Los Alamitos, CA 90720-1264.*

Joe Hootman

University of North Dakota

Software

Automatic memory manager

The most recent upgrade of the DOS memory optimizer, AT Last! Upper Memory Manager 6.0, automatically scans memory to find unused space. It then reboots the system and installs applications where they fit best. This version includes a compatible interface to install the DR-DOS 6.0 EMM386 driver. *RYBS Electronics; $80 (free upgrades).*

Reader Service No. 10

OS/2 backup

Nova Ware for OS/2 comprises a suite of programs for tape backup, conversion, and file transfer. A key feature, Nova Back, allows full or customized backup at regular times, even when the system is unattended. Nova Ware supports 1/4-inch tape drives (from 60 Mbytes to 1.35 Gbytes), 4-mm or 8-mm Exabyte cartridges, 1/2-inch nine track, and 3480-compatible tapes. Nova Back also supports stackers. *Nova Stor; $1,595 (Nova Ware for OS/2), $295 (Nova Back only).*

Reader Service No. 11

Window-based grammar checker

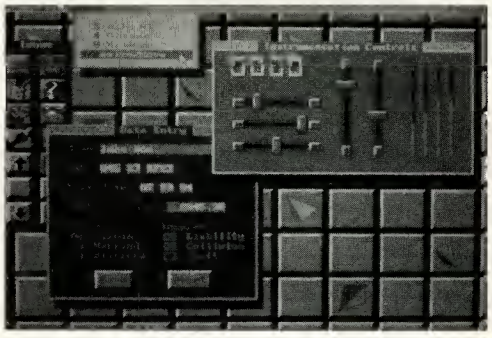
Grammatik 5 for Windows grammar-checking software launches from Windows applications for proofreading word processing documents, spreadsheets, databases, electronic mail, or other ASCII texts. The menu-driven program analyzes a root word's characteristics to determine how key words in the sentence or clause function. It then identifies structural or stylistic errors. Grammatik 5 requires a DOS machine with 80286 or faster microprocessor, Microsoft Windows 3.0 or 3.1, 2 Mbytes of RAM, and a hard drive. *Reference Software International; $99, $35 (upgrades).*

Reader Service No. 12

Library adds functions

C programmers can add windows, pop-up and pull-down menus, context-sensitive help systems, dialog boxes, icons, fonts, mouse support, and DOS support to their programs with Quick Windows Advanced tool kit. The kit's assembly language library operates without external graphics libraries or Microsoft Windows. Users can switch between text and graphics by changing the screen mode. Dynamic icons integrate into dialog systems and function as 3D command buttons, radio buttons, or check boxes. *Software Interphase; $149, $349 (with assembly source).*

Reader Service No. 13



Software Interphase's Quick Windows

Emulates DEC VT340

Teemtalk-340W, a DEC VT340 terminal emulator for PCs running Microsoft Windows 3.0, offers a variety of alphanumeric and graphics emulations. Menu-driven file transfer protocols include Kermit, X Modem, Y Modem, Y Modem Batch, and Modem 7. A scripting language allows users to automate file transfers, log-ons, and other procedures. Users can remap keyboard layouts. *Pericom; from $449.*

Reader Service No. 14

0272-1732/92/0600-0082$03.00 © 1992 IEEE

Imports data to Windows

The XVME-985 VME/DDE Server uses the Dynamic Data Exchange message-passing protocol of Microsoft Windows 3.0 to transport data from the VMEbus to selected Windows applications. Users can import and export data for Microsoft Excel and Wonderware's Intouch, via Xycom VME I/O and communications products. The server uses the DDE protocol to advise a Windows application of changes in data, format, and location. Users configure the server with pull-down menus and create a database of tag names and addresses. *Xycom; $500.*

Reader Service No. 15

PC terminal emulation

Users can create PC front-ends for complicated or unfriendly host systems with Trans Portal PRO, a data exchange tool kit. Applications can retrieve host data or update host applications in real time. The system includes logic for screen handling, field editing, validation, error processing, and help screen management. Trans Portal PRO works with dBase products and Microsoft Windows 3.0. *The Frustum Group; $1,495.*

Reader Service No. 16

Design software

PCB system

Scicards Version 27 printed circuit board design software includes a gridless editor that provides intelligent push/shove with on-line design rule checking. Automatic test point generation and output support automatic testing of dense surface mount and through-hole board technology. The system allows users to specify and automatically transfer the design rules and other parameters for complex designs. *Harris Scientific Calculations; from $45,000.*

Reader Service No. 17

Generates VHDL code

Express V-HDL, a graphical behavioral modeling tool, allows hardware engineers to design with the Statecharts graphic language. Engineers can analyze, validate, and revise models before committing to CAE simulation and synthesis. After verification, Express V-HDL automatically generates the equivalent VHDL and Verilog code. *i-Logix; price not given.*

Reader Service No. 18

Polygon editing

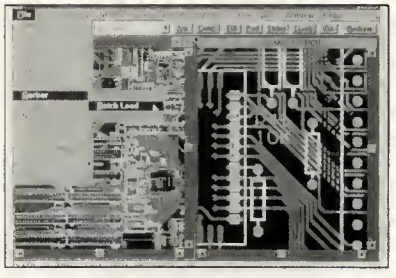
Tango-PCB and Tango-PCB Plus feature a polygon-editing command that lets users change the polygon shape by adding, stretching, and moving vertices. Other features of version 2.1 include faster redraw than in previous versions, a compression feature that reduces disk space by up to 50 percent, block operations allowing items to move from one layer to another, and facilitated metric conversion. *Accel Technologies; from $595.*

Reader Service No. 19

EDA on Windows

Advanced Pack, an electronics-design automation package for Windows 3.0 environments, combines the company's Advanced PCB design tools with autoplacement tools and a 16-layer, rip-up and retry autorouter. Other features include a global editing system, WYSIWYG print/plot automation, pen plotting, and a global, interactive AI-based autoplacement system. *Protel Technology; $2,990; free demo disk.*

Reader Service No. 20

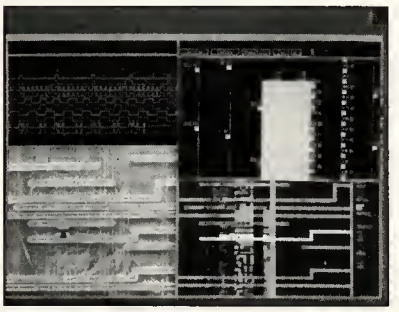


Protel Technology's Advanced Pack

Aids e-beam fault analysis

The Integrated Diagnostic Assistant, a semiautomated environment for e-beam and mechanical-probing systems, allows a probe operator to diagnose devices with CAD and CAE data. IDA's features include a guided probe algorithm that directs the user through the circuit and a controller that handles all communication with the simulator, even if it resides on another machine in the network. The system functions as an add-on to an existing e-beam prober or as a complete diagnostic system. *Schlumberger Technologies; from $125,000.*

Reader Service No. 21



Schlumberger's Integrated Diagnostic

Fuzzy logic design

Designers can simulate numerical algorithms, sequential logic, and fuzzy logic rules in the integrated design environment of RT/Fuzzy. The modular extension to the company's family of system design tools lets users work on complex dynamic control problems in a graphical environment combining rule-based reasoning with extensive numerical computations. Fuzzy logic rules are written in conventional if-then form. RT/Fuzzy Supports Sun-4, Sparcstations, VAX, and Hewlett-Packard workstations. *Integrated Systems; from $5,000.*

Reader Service No. 22

Communications software and hardware

Powers up host from remote

Remote Power On/Off installs in line between a phone outlet and a host modem and powers up a computer

when a remote caller signals. During the power-up process, the host's autoexec.bat file loads the right communication or remote control application. The unit eliminates the need to keep a host computer running to receive remote modem calls. Remote Power On/Off is compatible with DOS machines, Macintoshes, or Unix PCs. *Server Technology; $169.*

Reader Service No. 23

Combination Ethernet boards

Three 16-bit Ethernet boards, each with three-in-one Ethernet capabilities, support thick, thin, and 10Base T cabling. The CN650E uses programmed I/O memory access. The CN850E uses dual-port, shared memory access. The CN210E has bus mastering capabilities. Each board comes with software drivers for Net Ware 286 and 386, LAN Manager, OS/2 LAN Server, 3+ Open, PC/TCP, Xenix/Unix, and other network operating systems. *CNet Technology; $199 (CN210E), $289 (CN650E and CN850E).*

Reader Service No. 24

Token Ring drivers for OS/2

A set of software drivers for the Irmatrac Token Ring Adapter support LAN Server Version 1.3, OS/2 Extended Edition Version 1.3, and Novell Net Ware. Irmatrac users with OS/2 workstations can use the Novell Net Ware OS/2 ODI drivers to run OS/2 on a Net Ware LAN. The OS/2 EE drivers support shared network resources and a variety of host environments. *Digital Communications Associates; $895 (Novell Net Ware OS/2 ODI drivers free to Irmatrac users on DCA's bulletin board).*

Reader Service No. 25

Transputer-to-computer links

The Inmos IMS B300 Ethernet-to-transputer gateway connects a transputer subsystem to up to four computers, workstations, or PCs. Users on Sun, VAX/VMS, or DOS machines can access the transputer over an Ethernet LAN running TCP/IP.

The Inmos IMS B431 supports transputer links to an Ethernet interface via IEEE 802.3 LANs. The 2 × 3.5-inch module integrates a 16-bit transputer, 64 Kbytes of SRAM, and the company's Lance Ethernet chip set. *SGS-Thomson Microelectronics; $5,032 (B300), $980 (B431).*

Reader Service No. 26

Token Ring hub with 16 ports

A modular Token Ring hub, the Amptrac 16 concentrator, configures with network nonmanagement, management of the physical layer only, or management of the physical and media access control layers. The concentrator uses communications outlet inserts for port-level multimedia modularity with automatic internal impedance matching and filtering. Each unit supports up to 13 lobe expansion modules (260 shielded twisted-pair nodes). A distributed management and power supply architecture provides fault tolerance in the event of a management module or power supply failure. Amptrac sells with eight or 16 ports. *AMP, Inc.; from $112 per port.*

Reader Service No. 27

AMP's Amptrac 16

Access for the disabled

Adapta-Lan includes nine software packages to make computer access easier for disabled network users. The programs' capabilities include a screen magnifier, access for those with limited or no keyboard ability, telephone and modem access, and a visual indicator of the audio beep. *Microsystems Software; $2,995.*

Reader Service No. 28

Portable modem

Compouce Quadri, a modem for notebook, laptop, and other portable computers, weighs less than 3-1/2 ounces. Its features include four-speed, full-duplex communication and V21, V22, and V22bis for throughput rates up to 2,400 bps. A V23 mode offers videotex emulation, automatic call and response, Hayes compatibility, and an integrated RJ11 plug. The modem operates without batteries, consuming less than 10 mA through its serial junction. *PNB; $650.*

Reader Service No. 29

Modems up to 38,400 bps

The 9696 family of modems combine a 9,600-bps data modem with send/fax capabilities. All three support V.32, V.42, V.42bis, and MNP Level 5 standard protocols and can send data at 38,400 bps with V.42bis.

Self diagnostics ensure reliable operation. Features include CMOS technology, automatic dialing and answering, built-in speaker with software volume control, and RAM to store four phone numbers. A Macintosh version comes with Faxstuff, Quick Link II, and a Mac cable. *Logicode Technology; from $499.*

Reader Service No. 30

Logicode's 9696 modem

Signal processing software and hardware

Filter for real-time signals

The M27HC68 operates at up to 40 MHz to support HDTV, IDTV, video conferencing, and other applications

that need real-time video signals. The filter function is implemented as a transposed configuration with the input sample applied to 16-EPROM-256-word lookup tables simultaneously. Table outputs transfer in parallel to the arithmetic unit, which contains a chain of thirty-two 22-bit adders and sixty-four 22-bit-wide registers. The 1.2-µm CMOS E4 chip comes in 30- and 40-MHz versions. *SGS-Thomson Microelectronics; $59 (30 MHz, 1,000s).*

Reader Service No. 31

DSP processes 16.6 MIPS

The ADSP-2101-66 digital signal processor achieves a benchmark of 2.23 ms for a complex 1,024-point fast Fourier transform. The chip cycles in 60 ns and processes 16.6 MIPS. It is pin- and code-compatible with the manufacturer's ADSP-2105. Other versions of the chip include the 2101-50, which achieves 2.97 ms on the 1,024-point FFT, and the 2101-40, which achieves 3.72 ms. *Analog Devices; $61 (1,000s).*

Reader Service No. 32

DSP filter design on Sun 4

QE Design 1000+, a DSP filter for Sun-4 Sparcstations, performs complex mathematical computations for filter design and generates graphical displays and design reports. The chip is available under Open Look (X-Windows) and Sunview windowing systems and supports FIR, IIR, and Equiripple finite impulse response (Parks McClellan) filters. It uses a 64-bit floating point for all calculations. *Momentum Data Systems; $4,200.*

Reader Service No. 33

I/O board tutorial

A menu-driven interface guides Direct View users in setting up their I/O boards and explains options, including address, DMA channel, and interrupt level selection. After configuration, the program can be used to perform data acquisition on all channels, expressing the data as counts, volts, temperature, or strain. Tutorials explain the proper use of thennocouples and strain gauges. Block diagrams explain proper connection of field wiring to the I/O board's screw terminal panel. *ADAC; free with Direct Connect I/O board.*

Reader Service No. 34

Ten-bit ADC

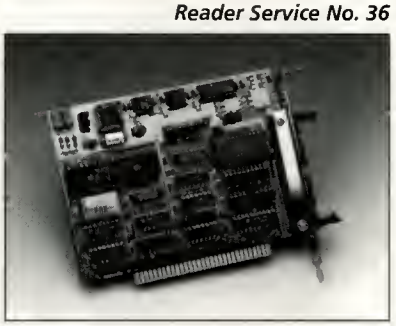
Designers implementing analog-to-digital circuitry in DSP and microprocessor peripheral applications can use the first in a planned series of 10-bit ADCs. The TLC1550IFN successive approximation register ADC supports high-performance systems such as cellular telephones and hard-disk drives. Its 10-bit bus requires only one read instruction for conversions. A three-state parallel port supports direct interface with most DSP and microprocessor system ports. Built in a 1-µm CMOS process, the device can access data at 35 ns and disable at 30 ns. *Texas Instruments; $6.27 (1,000s).*

Reader Service No. 35

Converts volts to frequency

Using a voltage-to-frequency conversion technique, the VF900 digitizes analog signals in resolutions from 10 to 18 bits. At 18 bits, it discerns signals as low as 10 µV. The unit features four differential analog input channels, programmable gain, selectable input voltage ranges, 16 digital I/O lines, and 12-bit analog output. Its A/D conversion rate at 10 bits is 1 KHz; at 15 bits, 30 Hz; and at 18 bits, 4 Hz. The package includes a programming disk with example software and development routines in Turbo C, Turbo Pascal, and Quick Basic. *Real Time Devices; $495.*

Reader Service No. 36



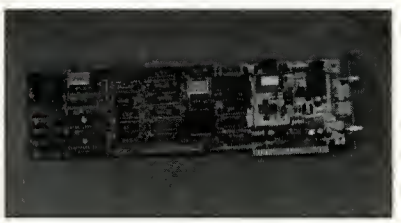
Real Time Devices' VF900

DSP development board

Engineers developing telecommunication DSP systems can take advantage of the TSP-C25, a single-slot IBM PC bus board. Based on Texas Instrument's TMS320C25 DSP, the board comes with 64 Kwords of zero-wait-state SRAM and 32 Kwords of EPROM with development libraries.

Other features include an FCC-compliant line telephone interface with a programmable 14-bit linear line codec and 2 Kbytes of dual-port SRAM. *DVP, Inc.; $1,895.*

Reader Service No. 37



DVP's TSP-C25

Links workstation to VMEbus

An adapter and support software combine to connect a Hewlett-Packard 700 Series Workstation to a VMEbus system. The Model 487 Adapter and Model 400-933 Support Software allow the workstation to act as a single-board bus master processor on the VMEbus system. A built-in DMA controller transfers data at up to 20 Mbytes/s. The software includes tools to support Unix read/write interface, interrupt handling, and atomic transaction emulation. *Bit 3 Computer; $600 (Model 400-933 Support Software), $2,850 (Model 487 Adapter).*

Reader Service No. 38

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 189 Medium 190 High 191

Product Summary

Joe Hootman
University of North Dakota

Manufacturer	Model	Comments	R.S.#
Chips			
Integrated Circuit Systems	ICS1700 controller	Quicksaver RISC-based integrated circuit controls the recharging of most nickel-cadmium batteries in 20 minutes then drops to maintenance mode using charge/discharge pulses at a C/30 rate. Recommended for notebook computers, cellular telephones, and other portable tools, the 16-pin DIP and 20-pin SOSM controller offers 0.5, 1, 2, and 4C charge rate capabilities through user-programmable select lines. $8.05 (10,000s, DIPs).	80
Texas Instruments	TL-SCSI285 voltage regulator	Fixed voltage regulator lets designers use the high-speed SCSI standard in battery-powered computers without power penalties. A 20-pin TSSOP version operates with a 0.775W dissipation rating at 25°C, has a lead spacing of 0.025 inches, and is 0.040 inches thick. Other versions include 14-pin DIPs and three-pin TO-220s. From $1.60 (1,000s).	81
Linear Technology	LT1116 comparator	This 12-ns-response device senses signals down to ground while operating from one +5V supply rail. Pin-compatible with the company's LT1016 comparator, the LT1116 interfaces directly to TTL logic with complementary outputs. The latch holds data as long as the latch pin remains high. $3.50 for plastic DIPS; $3.75 for SO-8 surface mounts. (100s).	82
Motorola	8HC16Y1 microcontrollers	Expanded 16-bit family combines a 48-Kbyte ROM and a RISC time-processing unit with 16 timer channels for applications requiring complex timing. In antilock braking systems, the Y1 independently monitors the spinning velocity of each wheel. An 8-bit, 68HC11-compatible CPU lets the 160-pin QFP Y1 perform control-oriented DSP functions. $38.69, initial beta samples.	83
Software			
Digital Communications Associates	IWM, v. 2.1.0 workstation	IRMA workstation for the Macintosh System 7.0 supports various Macintosh, IBM, or compatible mainframe communications, offering users Token Ring, coaxial, or LAN connections. Users can access System 7's Balloon Help and Apple's Publish features. $425; $95 (upgrades from Mac IRMAs).	84
Virtual Reality Laboratories	Vistapro	PC software produces various landscapes on screen for reproduction without copyright violation. The user sets a "camera" and a "target" position on a map so the virtual reality package can render a 3D, color painting-like view in minutes. Requires a 640-Kbyte RAM, a 3-Mbyte-free hard disk, VGA or Super VGA graphics card (VESA driver), and Microsoft-compatible mouse and driver. $129.95.	85

Manufacturer	Model	Comments	R.S.#
Systems			
Rapid Systems	R3700 pattern generator/logic analyzer	PC-based, 40-channel combination permits quick setup of a stimulus pattern and output in one-shot, continuous, or burst modes into the digital board under test. Features 2 Kbytes/channel data buffers, 32 data channels, four tristate channels, and one trigger channel. For external control the unit accepts clock, gate, tristate, and strobe signal inputs. $2,995.	86
SGS-Thomson Microelectronics	ST6210/15 starter kits	Designed for use with PC/AT or compatible computers, these 8-bit MCU kits helps users develop and evaluate applications. The starter kits contain a Basic programmer board; a flat cable link to the PC printer port and four EPROM-based microcontrollers; plus assembler, linker, simulator, and programmer interface software. Users may copy the kit's documented application software modules into their application software. $299 each; OEM discounts available.	87
Peripherals			
NMB Technologies	KB-3050 keyboard	Membrane keyboard for use with palmtop and notebook PCs measures 10.5-mm high, from key top to back plate. The 84-key design incorporates multiple load-bearing surfaces and lets users incorporate a mouse key. Its patent-pending (US, Japan) key switch includes three full-membrane sheets with a spring rubber dome. From $12 (OEM quantities).	88
Planar Systems	EL displays	Enhanced core product line features four thinner, lighter displays with 320×256- to 640×400-pixel resolutions. New features include variable contrast and brightness, an integral power supply, and a high-brightness capability. Typical power consumption in low-power mode is 6W for the 640×400 display. From $440 (100s).	89
Prima Storage Solutions	PDQ hard drives	External, transportable storage units expand PC, laptop, and notebook storage through the parallel port. Installation software detects interrupts, speed, and other port characteristics then installs a machine-specific driver with bidirectional capabilities in PS/2s, Compaqs, and Toshibas. Evaluation units support 85-, 120-, and 200-Mbyte hard drives and 44/88-Mbyte removable Syquest drives.	90
TDX Peripherals	TDX-348X cartridge drives	Half-inch cartridge technology supports the company's IBM 3480-compatible subsystem for the PC and workstations in one- or two-drive subsystems. The tabletop or rackmount models combine mainframe compatibility with 3-Mbyte/s performance, data interchange, and 800-Mbyte storage per cartridge. An optional 10-cartridge autoloader operates sequentially or can be randomly accessed.	91
Tecmar	Minivault 120, 250 tape subsystems	External 1/4-inch DC2000 tape backup systems support personal workstations and small LAN environments with up to 120- and 250-Mbytes of storage capacity. Accompanying software features automatic software installation, hardware configuration, and drive testing. From $539.	92

FOR DISPLAY ADVERTISING INFORMATION, CONTACT:

District Manager: D. Rodney Brooks; Tel: (415) 905-0260; Fax: (415) 896-1512.

Recruitment and Classified Advertising: D. Rodney Brooks; Tel: (415) 905-0260; Fax (415) 896-1512.

Director of Sales: Randall L. Stickrod, 544 Second St., Suite 200, San Francisco, CA 94107; Tel: (415) 905-0260; Fax: (415) 896-1512.

For production information, conference, and classified advertising, contact Heidi Rex or Marian Tibayan.

IEEE MICRO, 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1264; phone (714) 821-8380; fax (714) 821-4010.

IEEE COMPUTER SOCIETY®

A member society of the
Institute of Electrical and Electronics Engineers, Inc.

THE FOLLOWING INFORMATION IS AVAILABLE:

Contact the Publications Office;
to facilitate handling, please request by number.

- Membership application, student #203, others #202
- Periodicals subscription form for individuals #200
- Periodicals subscription form for organizations #199
- Publications catalog #201
- Compmail electronic mail brochure #194
- Technical committee list/application #197
- Chapters lists, start-up procedures #193
- Student scholarship information #192
- Volunteer leaders/staff directory #196
- IEEE senior member grade application #204
  (requires ten years practice and significant performance in five of those ten)

To check membership status or report a change of address, call the IEEE toll-free number, 1-800-678-4333. Direct all other Computer Society-related questions to the Publications Office.

PURPOSE

The IEEE Computer Society advances the theory and practice of computer science and engineering, promotes the exchange of technical information among 100,000 members worldwide, and provides a wide range of services to members and nonmembers.

MEMBERSHIP

Members receive the acclaimed monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

PUBLICATIONS AND ACTIVITIES

Computer. An authoritative, easy-to-read magazine containing tutorial and in-depth articles on topics across the computer field, plus news, conferences, calendar, interviews, and product reviews.

Periodicals. The society publishes seven magazines and five research transactions. Refer to membership application or request information as noted at left.

Conference Proceedings, Tutorial Texts, Standards Documents. The Computer Society Press publishes more than 100 titles every year.

Standards Working Groups. More than 100 of these groups produce IEEE standards used throughout the industrial world.

Technical Committees. More than 30 TCs publish newsletters, provide interaction with peers in specialty areas, and directly influence standards, conferences, and education.

Conferences/Education. The society holds about 100 conferences each year and sponsors many educational activities, including computing science accreditation.

Chapters. Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

OMBUDSMAN

Members experiencing problems — magazine delivery, membership status, or unresolved complaints — may write to the ombudsman at the Publications Office.

IEEE MICRO

Editorial Calendar

AUGUST 1992

European industry

- Recent developments in integrated circuit and microsystem technology from major European manufacturers

Ad closing date: July 1

OCTOBER 1992

Processing hardware for video communication

- ICs for HDTV compression
- ICs for HDTV communication
- Parallel processing systems with real-time video compression capabilities
- Multimedia systems with real-time video compression capabilities

Ad closing date: September 1

DECEMBER 1992

Special signal processors

- Recent information from the vital area of digital signal processors
- News about other techniques for signal processing
- Mixed analog/digital processors solve a real need
- Neural networks process signals following methods used by the human brain

Ad closing date: November 1

FEBRUARY 1993

Automotive/traffic microelectronics

- Worldwide developments in microelectronics for traffic and driving assistance
- Improving traffic safety with electronics
- Latest developments from Japan, the European Prometheus, and US IVHS programs

Ad closing date: January 2

APRIL 1993

Advanced packing and interconnect technology

- Critical packaging trends and issues
- Substrate and package technologies—for example, flexible, glass, or diamond substrates; few-chip or 3D packaging
- Attachment, bonding, and connection technologies, including five-pitch surface mount, laser applications, known-good die, and interconnection trade-off analysis

Ad closing date: March 1

JUNE 1993

Hot Chips IV

- This extremely popular issue presents the latest developments in microprocessor and chip technology used to construct high-performance workstations and systems as presented at the annual IEEE Computer Society TCMM-sponsored symposium

Ad closing date: May 1

IEEE Micro helps designers and users of microprocessor and microcomputer systems explore the latest technologies to achieve business and research objectives. Feature articles in *IEEE Micro* reflect original works relating to the design, performance, or application of microprocessors and microcomputers. All manuscripts are subject to a peer-review process consistent with professional-level technical publications. *IEEE Micro* is a bimonthly publication of the IEEE Computer Society.

Advertising information: Contact Marian Tibayan, Advertising Department, IEEE Computer Society, PO Box 3014, Los Alamitos, CA 90720-1264; (714) 821-8380; fax (714) 821-4010.

Articles may change. Please contact editor to confirm.